TAMPERE UNIVERSITY OF TECHNOLOGY

ATTE HYNNINEN

GEOGRAPHICALLY DISTRIBUTED SCALABLE WEB APPLICATION ARCHITECTURE

Master's thesis

Examiner: Professor Tarja Systä
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 5 May 2011.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
HYNNINEN, ATTE: Maantieteellisesti hajautettu skaalautuva Web-sovelluksen arkkitehtuuri
Diplomityö, 44 sivua
Syyskuu 2011
Pääaine: Ohjelmistotuotanto
Tarkastaja: Professori Tarja Systä
Avainsanat: Web, Web-arkkitehtuuri, Web-sovellus, skaalautuvuus, maantieteellisesti hajautettu, sisällönjakeluverkko

Diplomityössä suunnitellaan maantieteellisesti hajautettu skaalautuva arkkitehtuuriratkaisu. Arkkitehtuuri suunnitellaan verkkopohjaiselle kieltenopiskelupalvelu WordDivelle. WordDivella on käyttäjiä yli kuudessakymmenessä maassa. WordDiven nykyinen palvelin sijaitsee Suomessa ja se ei kykene tarjoamaan palvelua kaikille käyttäjille ympäri maapalloa samoilla vasteajoilla. Uudeksi arkkitehtuuriksi suunnitellaan maantieteellisesti hajautettu skaalautuva arkkitehtuuri, joka pystyy skaalautumaan joustavasti, kun käyttäjien määrä kasvaa. Skaalautuvuus halutaan tehdä näin, jotta arkkitehtuurin kehittyminen tapahtuisi kokoajan taloudellisesti tehokkaasti.

Web-sovellusarkkitehtuurit koostuvat ratkaisuista, jotka luovat verkkoarkkitehtuurin sekä ohjelmistoarkkitehtuurin järjestelmälle. Fyysinen arkkitehtuuri voi koostua yhdestä tietokoneesta tai jopa tuhansien tietokoneiden verkosta. Arkkitehtuuria voidaan skaalata lisäämällä tietokoneita yrityksen omaan palvelinkeskukseen tai yritys voi päättää ulkoistaa nämä arkkitehtuuriin liittyvät tekniset ratkaisut. Skaalautuvien arkkitehtuurien taustalla toimii yleensä hajautettu tietokonejärjestelmä, joka tarkoittaa sitä että kuorma on jaettu monen eri tietokoneen kesken. Tapa miten kuorma jaetaan voidaan tehdä monin eri tavoin. Pilvilaskenta tarjoaa yhden ratkaisumallin tämän toteuttamiseksi.

WordDive palvellaan tällä hetkellä yhdeltä fyysiseltä palvelimelta. Tärkein vaatimus uudelle arkkitehtuurille on kyky tarjota sama matala vasteaika kaikille käyttäjille. Suurin osa WordDiven liikenteestä koostuu multimedia-sisällön ja ohjelmistokirjastojen siirtämisestä käyttäjien tietokoneille. Tämän ansiosta on mahdollista muokata arkkitehtuuria siten, että siihen lisätään sisällönjakeluverkko (Content Delivery Network), jonka avulla pystytään pienentämään kaukana Suomesta olevien käyttäjien vasteaikoja. Tämän lisäksi WordDiven nykyinen palvelu voi tämän muutoksen jälkeen keskittyä niin sanottujen back end -toiminnallisuuksien ajamiseen. Arkkitehtuuria voidaan skaalata vielä enemmän hajauttamalla yhden palvelimen ratkaisu useamman palvelimen välille tai siirtämällä koko arkkitehtuuri pilvipalveluun.

Testaus suoritettiin lisäämällä sisällönjakeluverkko WordDiven testausympäristöön. Testauksia tehtiin kuudesta eri paikasta ympäri maapalloa. Tuloksissa oli jonkin verran hajontaa, mutta pääosin testit osoittivat selkeitä parannuksia vasteajoissa.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
HYNNINEN, ATTE: Geographically distributed scalable Web application architecture
Master of Science Thesis, 44 pages
September 2011
Major: Software Engineering
Examiner: Professor Tarja Systä
Keywords: Web, application, architecture, scalability, geographical, distributivity

A geographically scalable Web application architecture is designed in this master's thesis. The architecture is designed for a Web-based language learning service called WordDive. WordDive has a global user base with users in over 60 countries. WordDive's current Web server is located in Finland and it is unable to serve the content to the users around the world with equal response times. A new solution is sought by designing a geographically distributed scalable architecture that can grow flexibly as the amount of users increases. This is done to ensure the cost efficiency of the growing process.

Web application architectures consist of solutions that create the network and the software architecture for the system. Hardware architecture for a Web application can consist from a single computer or from a network of many computers. The architecture can be scaled up by adding more computers to the company's own data centre or the company can decide to use a service that does the scaling for them. The underlying technology behind scalable architectures is usually distributed computing, which means that the load is divided between several different computers. The division can be done in different ways. Cloud computing provides one solution model that can be used to implement this.

WordDive is currently running on a single physical server. The main requirement for the new architecture is to keep the response times low for all of the users. Most of the traffic while using WordDive is caused by the multimedia content and the code libraries that are transmitted to the user's computer. Due to this, it is possible to scale the architecture by adding a content delivery network, which will serve most of the content. The content is served from geographically distributed locations, which will decrease the response times on users who are far away from Finland. This also allows WordDive's current server to focus on backend activities. The architecture can be further scaled up via the means of distributed computing or by moving the entire architecture to a cloud-based environment.

The tests were done by adding a content delivery network to WordDive's test environment. Six different locations were used to do global testing. There was some deviation in the test results, but overall the test results showed clear improvement in global response times.

PREFACE

Writing this master's thesis gave me a chance to deepen my understanding on a subject that I have long wondered about. The writing process was quite natural due to my passion towards the subject. After reading hundreds of pages of theory, it was a pleasure to actually get my hands dirty and have a try at actually doing something practical and to see how these things really work. I would like to thank Professor Tarja Systä for the excellent help she gave to me. In addition Timo-Pekka Leinonen and Juha Rintala from WordDive and Tommi Palm from Nokia deserve equal praise for their assistance. And above all I would like to thank my family, my friends and my girlfriend Emmi for their constant support throughout the writing process.

In Tampere on 19.9.2011

Atte Hynninen

CONTENTS

TERMS AND DEFINITIONS

API	Application Programming Interface, a software interface, which provides a certain set of rules on how to interact with it.
Cookie	A data file made by a Web server to save data on the client computer.
CPU	Central processing unit, the part of a computer system that handles the instructions set by a computer program.
DNS	Domain name system, which translates domain names to IP-addresses.
HTML	Hypertext Mark-up Language, a mark-up language for building Web pages.
HTTP	Hypertext Transfer Protocol, a protocol for transmitting data over the Internet.
LAMP	A solution stack of open source software for a viable general-purpose Web server. The acronym's letters come from Linux, Apache, MySQL and Perl/PHP/Python.
RAM	Random-access memory, a form of computer data storage.
TCP	Transmission Control Protocol, one of the core protocols making it possible for transmitting data over the Internet.
WYSIWYG editor	What You See Is What You Get, a type of a graphical user interface editor that previews the end result immediately.
OS	Operating system, software that handles the computer's hardware and provides services for programs to use.
OSI model	Open System Interconnection model, a way of dividing communications system into layers.

1 INTRODUCTION

This master's thesis focuses on a scalable Web application architecture in a geographically distributed environment. The architecture is designed for a Web-based language learning service WordDive, which currently has users in over 60 countries worldwide. Because of the growing amount of users and the poor latency for users from faraway, a scalable architecture is needed. The thesis looks into existing solutions and builds on these to create a solution that fits into this environment. During the research, an expert interview was done to help in finding out a suitable solution for this environment. Based on the findings, an architectural design was made and evaluated.

WordDive's current architecture is a simple system consisting of a single server, which runs the web server and database server software. This is a good and an affordable solution for start-ups, but when the load on the server keeps on increasing a scalable solution needs to be sought. The current approach of scaling up by upgrading the server to an always-faster one will not suffice, when the amount of users reaches a certain limit. This kind of an approach will also not help with customers that are using the service from the other side of the globe as the problem is due to the distance between the server and the user. This thesis explains how an architecture like this can be scaled up from handling thousands of users to all the way to millions of users, so that the scaling up is always done at the right time in a cost efficient way.

This thesis solves a common problem among start-up Web companies, where the amount of users keeps on increasing and they come from further away than before. To provide the same user experience to all of these users, a scalable Web architecture needs to be made. In many cases a website is launched with focus to a single market, but suddenly the website starts to get noticed on different sides of the globe. Because the website has only been designed for that single market and its users, the user experience might not be as good for those users from different places. Out of this comes a need for internalisation, but also a technical need for providing better response times to the large mass of people using the website from faraway.

The solution that is presented is a generic one that can be used by developers who are faced with similar difficulties. It should still be kept in mind that the solution may not be good for all websites, but rather a solution that fits this case environment and any other website that is in a similar situation. After reading this thesis the reader should have a broad understanding of Web architectures in general.

In the solution a content delivery network is added to the current architecture. A content delivery network can significantly decrease Web application's global response times and also decrease the load on the Web server, which is used for serving the Web

application. Content delivery networks scale automatically, so that the developers do not have to worry about having enough server resources to serve the content. A content delivery network may not be the final architecture for WordDive, but instead it is a cost efficient solution for the current situation.

Chapter 1 gives a brief introduction to this thesis. Chapter 2 explains the theory behind geographically distributed Web application architectures. The case environment is presented and analysed in Chapter 3. Chapter 4 describes the model of the new architecture and provides further options for scaling up. Testing is described, the test results are presented and finally analysed in Chapter 5. Chapter 6 discusses the final conclusions.

2 THEORY

In this chapter, we explain the theory behind scalable Web application architectures, focusing on subjects that relate especially to scalability. The chapter also explains general theory about Web application architectures.

2.1 General concepts

2.1.1 Scalability

Schlossnagle (2006) defines scalability as the ability to make a solution that solves and fits the problem as the scope of the problem increases. In Web architectures scalability is often described in two ways: vertical and horizontal scalability. As described by Allspaw (2008, pp. 19-20.) vertical scaling architectures are architectures that are scaling up by increasing the resources internally to a server, whereas horizontal scaling architectures are scaling up by adding more similarly functioning nodes to the existing infrastructure i.e. more servers. Horizontal scaling is seen as the better solution in Web applications as even though vertically scaling up may be the easiest choice at first, at some point the costs will rise dramatically, relying on a single computer also introduces the risk of a single point of failure. It is also important to understand that as computers become faster also horizontally scaled architectures should be vertically scaled by upgrading the hardware, as this can be more cost effective than simply continuing to scale horizontally. Horizontal scaling also has its own share of problems when it comes to synchronising the different nodes.

2.1.2 Capacity planning

To understand the effects of a new architecture one must first measure how the system is operating currently. Allspaw (2008, pp.24-25.) explains the basic idea of monitoring tools as a set of tools that gather statistics over a certain of period of time. The tools can also possibly provide graphs that will illustrate the progress of these statistics. The statistics can for example be about the CPU, RAM, network or disk usage. It is also important to understand the source of this usage. For example, if the database and the Web server are running on the same hardware, it is important to understand, which piece of software causes the usage. One should also not forget that even the metric collection system can affect the system's behaviour. Figure 2.1 shows the basic architecture of a metric collection system.

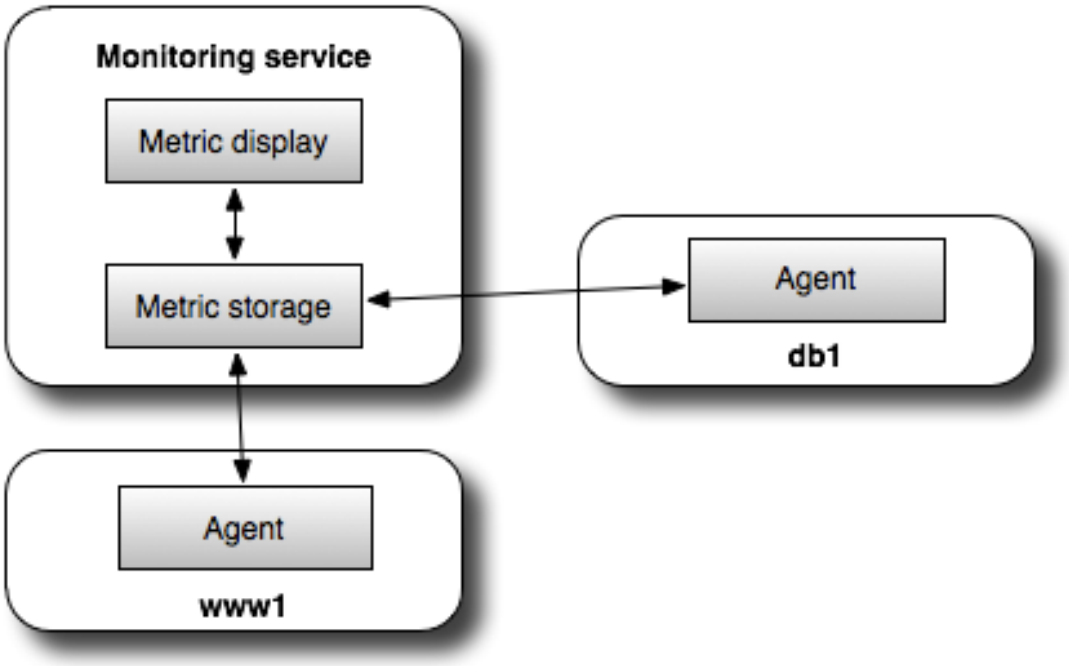

Figure 2.1 The basic architecture of a metric collection system, the figure is based on the figure from Allspaw(2008,p. 26)

Understanding when to scale-up is basically a numbers game, it is not economically feasible to scale-up too early and the user experience gets worse if it is done too late. This is why the process should be planned according to the monitored data. When the tools are set-up, the data should be monitored, and trends should be sought for. This will allow for finding the bottlenecks of different servers. To safely find the bottlenecks and to understand the limitations of the servers, the bottlenecks should be manually tested. It is also important to understand what is the bottleneck on the server, the CPU, RAM, disk capacity, network access or something else. Each server's work should be optimised for the task they are doing. For example, if a server that is meant to do background statistics processing is using a lot of memory but only a little bit of its CPU, it might be reasonable to add a another CPU heavy task for this server, thus using that computer's full potential. This will allow for using the full potential of the servers and thus reducing any redundant server usage. The server usage should be set to as high percentage as possible, keeping in mind any fluctuations in the data with an appropriate margin of safety. One other thing to keep in mind is the server's tasks' sensitivity, meaning that servers with non-critical tasks can be run on higher loads. (Allspaw. 2008, pp.63-91.)

2.2 Hosting

A Web application can be hosted in a lot of different ways. This section discusses traditional ways of hosting a Web application.

As Henderson (2006, pp. 15-19) explains, a company can decide to host their Web application by themselves, which although maybe an affordable option, but may not be reasonable at all on a small scale due to possible problems of having servers in an office environment. Problems may arise, e.g. network connection problems caused by the other people living in the same building. Other options include renting the hardware as a service from a Web-hosting provider, where the level of commitment to the hardware can be decided on. For example, you could decide to use shared hardware, meaning that your software would run on the same hardware as some other company's software. A step up from this is dedicated hardware, so that the hardware that is used will only run your software. The level of how you can control this piece of hardware or the OS may vary. One possible option is also to co-locate the hardware, meaning that some 3rd party company will provide you with the facilities and the services related to it, but you will be responsible for getting the hardware that you are using. But as Henderson mentions, special care should be taken when choosing the co-location provider as a bad one can cause a lot of problems. Finally if a company gets to a point of having a few thousand servers it is usually beneficial for the company to build their own data centres. This is a huge task, as the company will have to build their own purpose-built facilities and hire people to take care of it. A constant surveillance of the data centre may become extremely expensive. For a long time there has existed some secrecy on how different companies build their data centres, but one option is to build data centres according to the specifications provided by Facebook's Open Compute project (2011), which aims to open up possibilities especially to companies located in poorer countries to build efficient data centres affordably.

2.3 Cloud computing

Cloud computing in a nutshell is packaging up computing resources into a service. This way the customer does not have to care about the infrastructure behind the service. Cloud services usually work on cost-per-usage basis. This makes the initial payment low for the customer compared to, e.g. building a data centre.

Mell & Grance (2009) present the National Institute of Standards and Technology (NIST) definition of cloud computing. It defines the essential characteristics for cloud computing. These features include *on-demand self-service*, which means that the user should be able to provision computing capabilities without any human interaction. Cloud computing services typically have *broad network access*, meaning that the hosted services should be broadly available, when used by standard mechanisms. This also means that the services should not be device or location dependent. *Resource pooling* means that a cloud computing service typically assigns the resources to the consumers according to consumer demand. It is also stated that the user does not have control of the exact location of the provided resources, but the user may be able to specify the location of the resources on a higher level, for example, the user can choose the country where the resources are located in. The fourth characteristic is *rapid elasticity*. This

characteristic describes that the service can be scaled up or down rapidly, so that it appears that there are no limitations. Finally the service should be a *measured service* so that it is able to measure and report the amount of usage. Measuring the usage of services makes it possible to charge the customers according to their consumption.

Systä (2010) also presents some other claimed characteristics for cloud computing. Reliability is often kept as an advantage that cloud-based systems have. This is because cloud-based systems use redundant sites and have the infrastructure available to recover from almost any kind of failure. However this is not the ultimate truth as many major cloud-computing services have suffered outages. Another claimed characteristic for cloud computing is the security that it provides. The security can get better due to the possibility of centralising all of the data into the cloud and due to possible security focused resources in the cloud environment. Then again cloud-based systems also take the control away from the user, which might be troublesome with sensitive data. Cloud-based systems are also often praised for the ease of maintenance, because cloud-computing applications do not have to be installed on each user's computer. This again makes the maintenance easier because the updates reach the clients immediately. However this does not guarantee improved maintenance.

Cloud computing is commonly divided into three different levels: cloud infrastructure i.e. *infrastructure as a service* (IaaS), platform clouds i.e. *platform as a service* (PaaS) and finally application clouds i.e. *software as a service* (SaaS). Allspaw (2008, p.109) defines the different cloud services in the following way.

2.3.1 Infrastructure as a service

These kinds of services most resemble a traditional Web hosting service. The user can subscribe to, for example, computing units that they can use. In often cases these kinds of services are built on top of virtual hardware infrastructure. The two main tasks of such a service are providing computing power and storage for the user. The service provider is responsible for housing, running and maintaining the hardware equipment. The service itself is a computing environment, where the users can run their own applications or services.

This kind of a service usually allows scaling-up by simply upgrading the service that the user has, e.g. the user can add a new server to their subscription or upgrade the server to a more powerful one. These kinds of services are flexible, but might need to be accompanied with other services that will handle the on-the-fly management of the servers, so that there is always an optimal amount of servers being used.

2.3.2 Platform as a service

Platform as a service means that the users are left with the ability to create and run their software on top of the provided platform, but the user cannot modify the actual platform, where the code is ran on. There can be elements such as databases as part of the platform that the users can use with their software. The users have to build their

software specifically for this platform. As Systä (2010) points out, the user does not have to care, for example, about the parallelism or distribution of the software as the software stack within the service itself takes care of it.

2.3.3 Software as a service

Software as a service is term for Web-based software that allows the customers to use a service in the Web. As explained by Systä (2010), these kinds of services are directly aimed at end users. This type of service can be a lot of things, for example, a Web-based email service or an enterprise resource planning (ERP) service.

2.4 Architectural solutions for scalable Web application architectures

2.4.1 Distributed computing

A computing system becomes a distributed computing system when it has two or more computers in it. A distributed system is usually needed when it becomes too expensive to vertically scale-up a single computer and so another computer is needed for dividing the load. The computers can be replicated copies of each other or they can be assigned for different purposes. If the application is supposed to be highly available it is important to make the architecture so that there is no single point of failure, this means that if a single machine breaks in the architecture, some other machine can take its place, preferably immediately.

Distributed databases

Henderson (2006) points out a common architecture model used when distributing databases. It is basically a *Master-Slave* model where there exists a *Master* computer that handles all of the write operations to the database and several *Slave* computers that will handle all of the read operations. This kind of an architecture will enable faster read speeds, but will not make it faster to write into the database. It is also important to understand that all of these servers will still write all of the operations to the databases, so the read speed's gain is only gotten from the fact that the read operations are now divided between the different computers. This is a good architecture for websites, where there are a lot of read operations, but not too many write operations, which is the case for example, in news websites. An architecture like this may work so that if the *Master* computer breaks down, one of the *Slave* computers becomes the *Master* and the website will continue to work. Figure 2.2 illustrates this kind of a *Master-Slave* architecture.

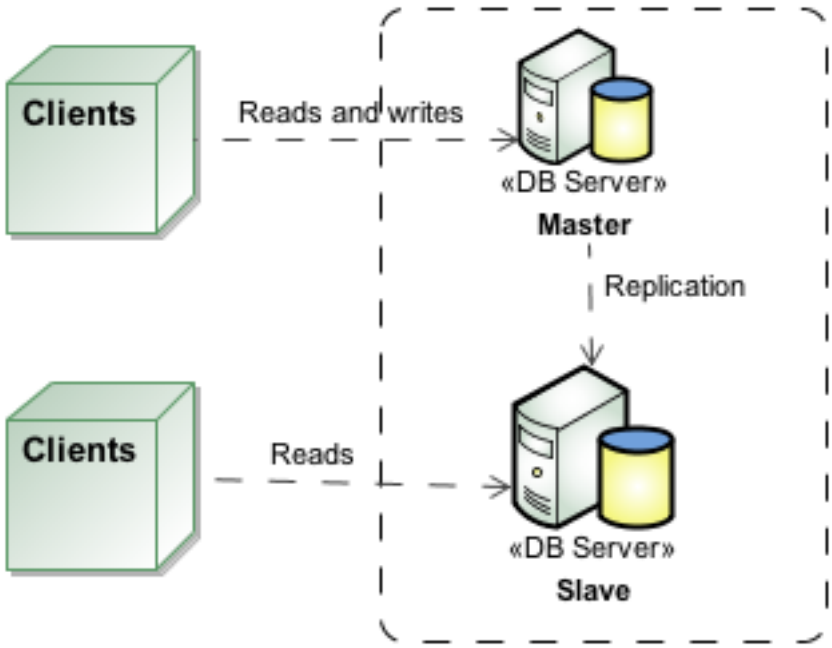

Figure 2.2 Master-slave database architecture, the figure is based on the figure from Henderson(2006)

To make it possible to scale write operations, Henderson (2006) explains that the databases will need to be partitioned. Partitioning helps to decrease the amount of writes that a single database will have to handle. Partitioning can be horizontal or vertical.

Vertical partitioning, which is also known as *clustering*, is a simple solution but also a limited one. Clustering the databases means that the database is divided into several databases running on different computers and each database will handle some specific tables from the overall database. This will also mean that once each computer serves only one table, the set of clusters can no longer be scaled anymore in this manner. Tables that have content, which is often joined in the application's code, should be kept in the same cluster. There are no benefits from using this solution if each cluster has to be used every time there is a request from the client. This would actually make the same amount of connections to each of the clusters than what would be made to a single database handling all of the database tables. This kind of a solution will also make the application's code more complicated, because the division to different databases needs to be made in the application code. (Henderson. 2006.)

Henderson (2006) describes *horizontal partitioning* or *federation* as the act of slicing up a single database table's data into some arbitrarily sized chunks. These chunks are usually called shards or cells, thus the term *database sharding*. This kind of a solution is also difficult with tables that have joins between them, as joins will have to be made between different shards and the correct shard will always have to be sought. To prevent this kind of a problem the shards should be designed so that cross-shard selects and joins should not have to be made. For example, all the data that a single user will use on the website should be saved in the same shard therefore minimizing the need for cross-shard selects or joins. To gain this kind of a structure, denormalisation of the

database's structure is often needed. To keep federation away from the application's code an interface can be made between the application logic and the databases that handle the federation. There are also network based database solutions, which can do the federation without changing the application logic. One this kind of solution is MySQL Cluster (2011). However as Davies & Fisk (2006, pp. 8-10) point out, MySQL Cluster comes with its own limitations and might require some changes to the database schema if such already exists.

Distributed web servers

Working with web servers is a bit easier than with database servers, because Web servers do not have to hold persistent data and the amount of modifications to the servers is relatively low. This means that the developers make the only modifications that will have to be made to these servers. The changes may be to the application logic or for example to the server configuration. As Schlossnagle (2006) points out, all of these modifications should be under version control, because tracking changes between several different servers can become extremely difficult and because version control also allows the changes to be changed back.

Perhaps the simplest way to spread the load is by spreading different features of the website to different systems. For example, if a website has an administrator side, moving this side to its own Web server, should lessen the load on the non-administrator side's server. However, this kind of distribution will require application level changes, which will further complicate the application's code. This kind of a system may also end up in situations where load is not equally balanced as some computers might be underutilised.

Another simple way to distribute the load is to use for example a load balancer that divides the load between several Web servers that are identical. The Web servers do not have to synchronise their data between themselves because any persistent data is saved to databases and any temporary data is used only on that single Web server. If the Web servers need to be updated, they can be updated at the same time, or one by one. The load balancer can be used to divert the traffic away from the Web server that is being updated. To decrease the possibility of configuration errors and to keep the system more simple, same hardware should be used on all of the computers.

2.4.2 Load balancing

A load balancer's main purpose is to distribute the load among the different machines in the system. Load balancers are most commonly installed on the front-end machines to divide the load on the Web servers, but they can also be used in the back-end for instance to spread load across databases. Load balancers handle the load distribution by using algorithms to figure out where the load should be directed. Load balancers can also be used as a tool when the production environment is changed, for example to direct users away from a server that is going to be removed. A similar kind of method

can be used to test new features on the Web application with real users, by just letting a portion of the users to access the new server with the feature. Load balancing may also cause problems if the used algorithm is calculating the load incorrectly e.g. if the algorithm trusts that the load is directly correlated to the amount of users on a server, which often might not be the case. (Allspaw. 2008, pp.24-25.)

As Henderson (2006) mentions, the easiest way to do load balancing between a couple of servers is to do it by the use of DNS servers. The basic idea is to list several different IP addresses for a single domain name, the DNS servers will then use these addresses and the clients will be forwarded to the different servers. As Henderson continues, this is not a good way of doing load balancing for many reasons. The biggest problem is the speed at which the DNS servers propagate the linking of IP addresses to domain names. If a server crashes, it can take up to several days until that server is removed from the DNS servers and therefore it will seem like the service is offline for those users that are pointed to that server. The other big problem is that the load is not necessarily balanced evenly and due to DNS caching; the users will be stuck on the same server for an hour or more.

However as Schlossnagle (2006) points out, DNS servers can use different routing methodologies and the use of Anycast methodology can provide better results, because the method works so that the client's request is forwarded to the closest server available. The server's distance to the user is determined in the network distance rather than the real geographical distance. This can also be combined with other methods to meet high availability requirements. There still exists the problem of non-equal load balancing, which could lead to a situation where for example the servers are overloaded in Europe and the North American servers have lots of free capacity.

To make the load balancing more balanced, a hardware or software based load balancer can be added to the infrastructure. The load balancing can be done on different levels. Traditionally load balancing has been done on a lower level, because the load balancing has had no use for more detailed information about the data that is being transmitted. To get more detailed data, a more detailed level can be used, which will allow e.g. load balancing according to the URL. The algorithms working inside the load balancers generally work along the same principles. They are following the information provided by the available servers and the information about previous and existing connections. One major issue with load balancers is the case of statelessness on the servers. For example in PHP, it is possible to have session based user specific data saved on the Web server, if such a user is then redirected to another server, this session data cannot be used unless this session data is saved in a common place that can be reached from both of these Web servers. Another approach is to configure the load balancer so that the same users are staying on the same Web server for the duration of the session, so this problem can be prevented, but this will also make it harder to keep the servers' loads perfectly balanced. The following figure shows the basic idea of load balancing. (Henderson. 2006.)

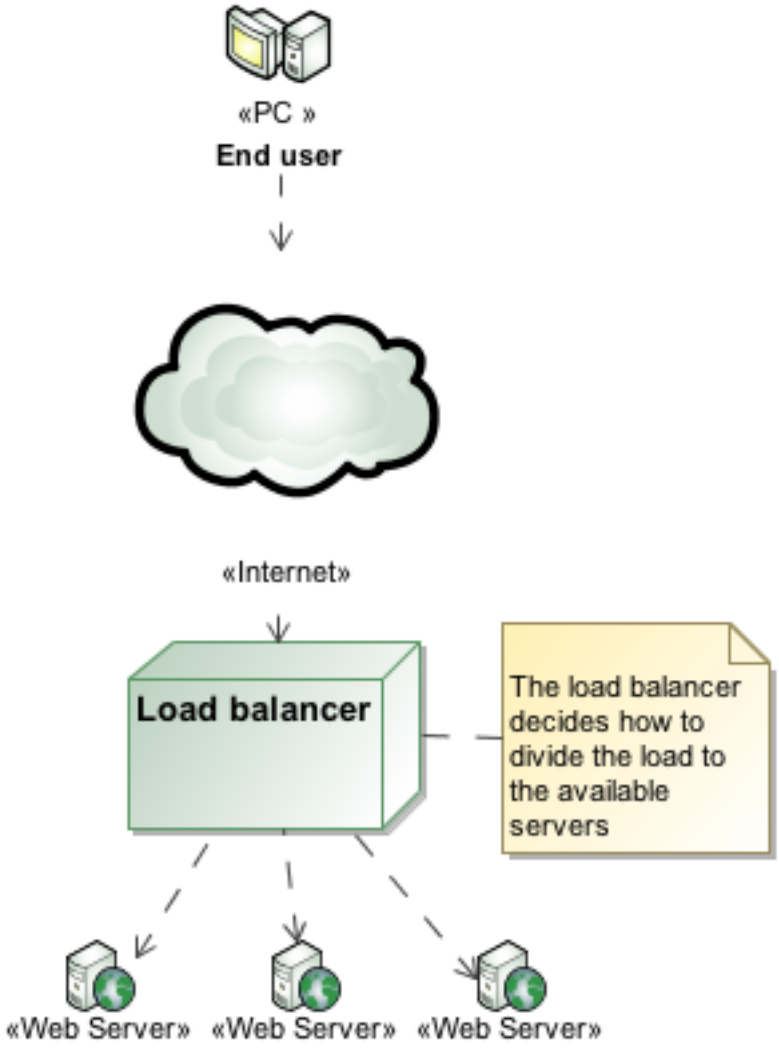

Figure 2.3 Basic idea of load balancing, the figure is based on the figure from Henderson(2006)

Schlossnagle (2006) states that it is better to look for other options than to use the technique of session stickiness, where each user is kept on the same server, because this creates too many problems with the load balancing. As the Web consists of short unforeseeable requests one after another, it means that bundling every user's request to one specific server creates quite a big problem and this will overly complicate the algorithms used for load balancing. Schlossnagle recommends that the data that is usually saved on the Web server should instead be divided to data that is being saved for example on a database that can be reached by any of the Web servers and the clients' local storage should be used as much as possible, which basically means that client side cookies should be used as much as possible.

2.4.3 Caching

Caching is a commonly used technique in a Web application architecture. The basic idea of caching is to copy data that is often requested to a place, which is faster to reach, thus saving time. The idea of caching is illustrated in Figure 2.4.

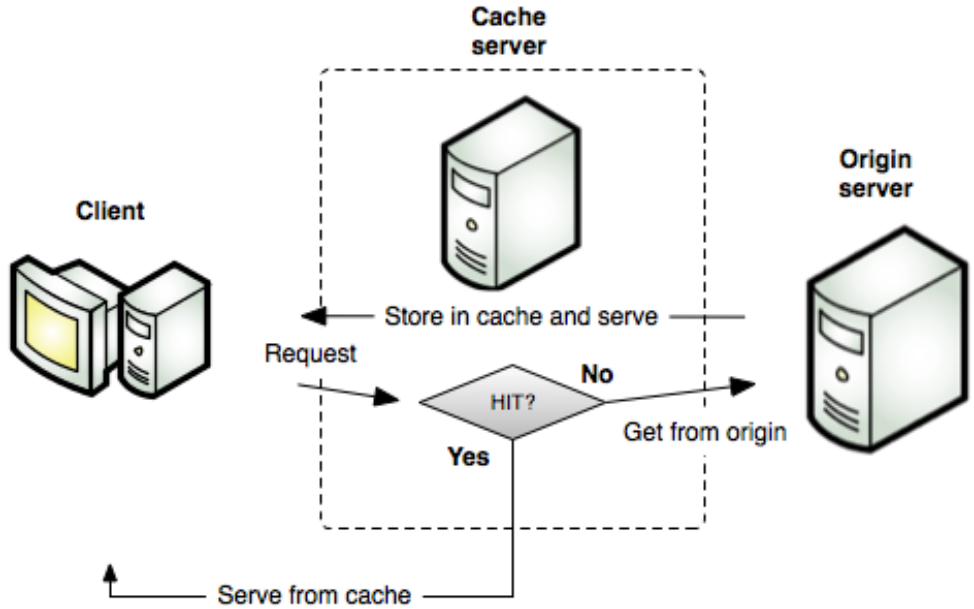

Figure 2.4 Basic caching mechanism, the figure is based on the figure from Allspaw (2008,p. 46)

As shown in Figure 2.4, the requested item is first sought from the cache server, and if it is not found, it is fetched from the origin server, saved to the cache server and served to the client. An efficient use of caching can decrease the response times and decrease the load on the servers. Caching can be done on the frontend machines i.e. the Web servers or in the backend, e.g. on the database machines. As caching does in fact copy data, special care needs to be made on keeping the data up to date. Therefore it is less effective on data that is subject to change all the time. As caches are usually smaller in size than the actual storage, all data cannot be held in the cache. This is why caching systems use algorithms to figure out, which data to keep in the cache. The most common algorithm is a simple Least Recently Used (LRU) algorithm, which is basically a stack of elements, where a requested element is picked up to the top of the stack and the last element is removed from the cache. (Allspaw. 2008, pp.45-48.)

Schlossnagle (2006) points out the importance of choosing the right style of caching for the right situation. For example different caching methods can be used for dynamic and static data. It should also not be forgotten that the closer that the cache is to the user the faster it is to fetch data from there. Some Internet service providers have caches of their own, which cache Web content that their customers are using. Obviously the closest cache to the user's Web browser is the Web browser's own cache, which helps out with reoccurring elements on the website.

2.4.4 Static content serving

As Schlossnagle (2006) points out, the content that is being transmitted between the client and the service is often static content and the methods for serving dynamic content are not typically ideal for serving static content. Static content is content on the website that stays the same, it can be for example an image on a website that is the same for all of the users and the users have no way of modifying it. Dynamic content is content that is constantly changing, for example a Web page can be dynamically created according to the user's preferences. This kind of customised content is different to different users. Since content like images usually also take a fair amount of disk space, transmitting it to the client can be slow. If a considerable amount of the data being transferred is static content, it might be reasonable to start serving this content independently from a server that is able to communicate faster with the client.

2.4.5 Content delivery networks

Buyya et al. (2008, pp. 3-5) explains how the idea of content delivery networks was built on the need for new kind of systems that could handle large unexpected growth on a website's traffic. A content delivery network (CDN) consists of a collection of mirrored web servers, called *edge servers*, which collaborate to deliver content to end-users. The aim is to provide a service that works effectively and transparently. The network is scattered across the globe therefore bringing the content closer to end-users, which in turn helps to deliver the content to the end-user faster and more reliably. The servers in the network are typically called edge servers. Figure 2.5 shows the basic idea of how a content delivery network works.

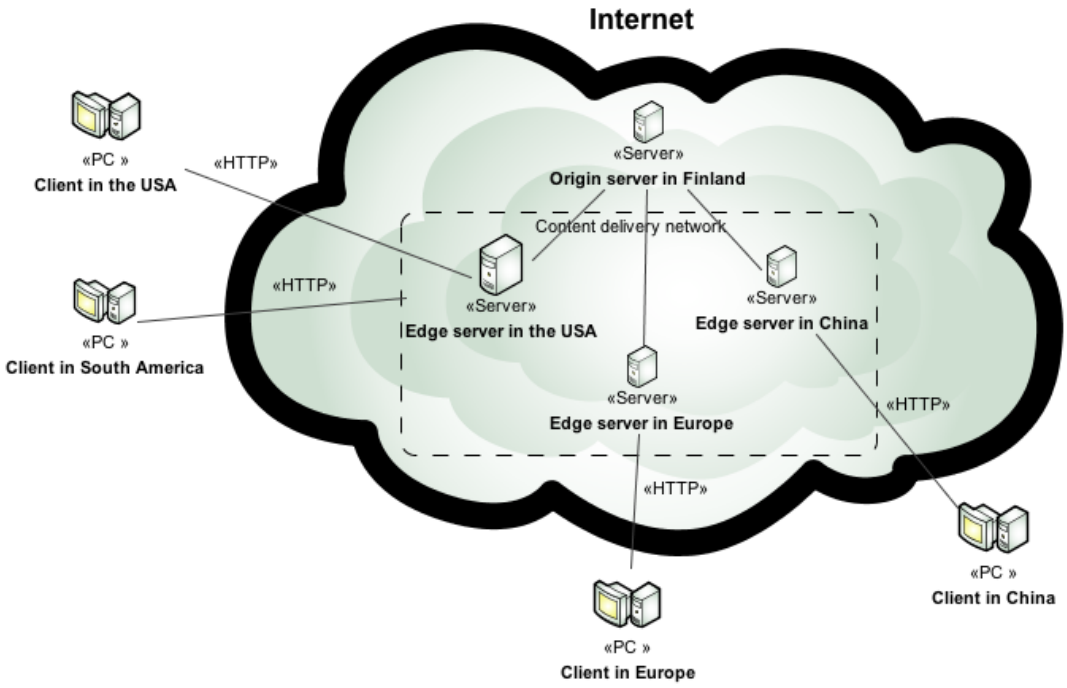

Figure 2.5 Content delivery network overview, figure based on the figure from Pallis & Vakali (2006, p. 102)

As shown in Figure 2.5, a content delivery network consists of edge servers and an origin server. The origin server does not necessarily have to be inside the content delivery network. The edge servers fetch the data that they are serving from the origin server and they share the data to the clients that are nearest to them. So, for example clients from Europe will end up requesting their files from the edge server in Europe and the clients from South America will get their content from the edge server in the USA, if no edge server is located nearer.

Content delivery networks typically have functionalities like request redirection and content delivery services, which means that the requests are directed to the closest suitable server in the content delivery network thus bypassing congestion. The content is typically replicated to the network through an origin server, which holds the master copy of the content. CDN services usually also provide some kind of a management service that can be used, for example, to monitor the traffic and to make changes to the network. Content delivery networks can be used for serving static and dynamic content. They are commonly used for example to serve streaming video or images, which require a lot of bandwidth. (Buyya et al. 2008, pp. 4-8)

3 CASE ENVIRONMENT

3.1 Description

Voctrainer Ltd is a young start up company building a Web-based language learning service WordDive. As the service is still quite young, so is the case environment. WordDive application has recently scaled up from a set of virtualized servers to a dedicated physical server on its Web-hosting provider. However, to further scale up, new solutions have to be sought.

WordDive provides the means for studying languages online through a Web interface. The user can choose between several languages which to study. The actual exercise is done by looking at an image, associating the image with a real world concept and trying to write it to the answer field. If the answer was correct the pronunciation of the word is played to the user.

The following Figure 3.1 illustrates the main exercising view from the WordDive application. The image that the user is learning is shown on the left, the answer is written to the textbox on top right. The user can also get some hints by translating the correct answer or by listening to the correct answer by pressing the corresponding buttons. In this example the user is exercising Spanish and is trying to learn the word "estar de pie" ("stand" in English). Figure 3.1 shows how the user has already written the correct answer to the textbox.

Figure 3.1 A screenshot illustrating the main exercising view from the WordDive application

The user's progress is tracked constantly and the software adapts to the way that the user is learning, so that the user will not be asked the same words too many times and so

that the speed of which new words are shown to the user is suitable for his/her learning rhythm.

3.2 Current system

The current system runs on a single high performance server located in Finland. The server is subscribed as a service from Nebula, a Finnish Web hosting company. The server runs a virtual computer that runs a Linux based operating system. By running the operating system on top of a virtual computer, the whole operating system can be easily moved to run on another physical computer in case of a hardware failure. On the operating system the most significant software are the virtual servers for serving the Web applications and the databases. The Web server is an Apache Web server and the database runs on the MySQL 5 software. In addition, PHP 5 software is used for enabling the backend functionality. The combination of these technologies creates a software bundle called LAMP, which is widely used in Web applications and as such is also widely portable. Figure 3.2 shows how the connection between the application and the end users is now done.

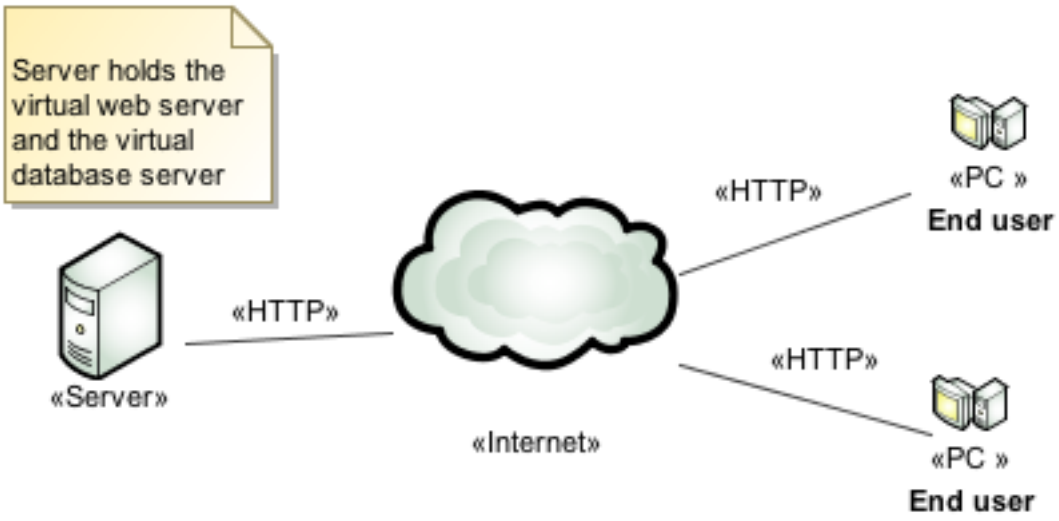

Figure 3.2 High level hardware diagram of the current system

As can be seen from Figure 3.2 the current system is simple. A single server in a single location provides the application for all of the end users. The server is constantly mirrored so an error in the hardware causes minimum downtime for the application. Currently all of the data is centralised on this one server, so no synchronisation has to be done with other servers.

As the application provides personalised behaviour for each user, the database's state is constantly changing. The structure of the database can be divided into data that is used by everybody and to data that is user specific. In addition to these, there is also data that is only being used by the software and data that is only for the developers. Figure 3.3 shows the overall structure of the database on a very abstract level.

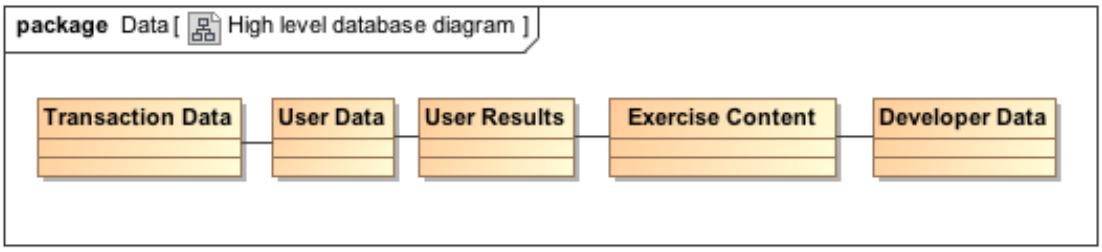

Figure 3.3 High level illustration of the current system's database's structure

In Figure 3.3 the "Transaction Data" refers to the data about the money transactions. This data is absolutely critical and should be centralised. The "User Data" component means data about the user, which for example can be used for identifying the user. This data is fairly persistent and does not update frequently. The "User Results" refers to the data that is collected when a user exercises a language in the service. This data is constantly changing. These first three components represent the user specific data that is changed by individual users. The common data component "Exercise Content" represents the data that is developed by the content developers. This is mainly textual data about the content that is being shown to the user when exercising a language, but this also includes metadata that is being used by the software. This common data is shown to all users. Finally, the "Developer Data" component is data that is being used only by the developers, so that they better understand what they are doing. This data is not shown to the consumers.

Even though WordDive is a Web 2.0 application, where the application is customised for the users and the users create their own data, the data that the users create is merely numerical data, so the application is not especially heavy on disk capacity compared to applications that collect images and videos from the users. However, the users still do create huge amounts of numerical data, which can be heavy to process on the backend machines.

3.3 Requirements for the new system

The main requirement is to keep the current level of user experience to all of these users even in the future when the amount of users keeps on growing, in this case this is done by keeping the latency even. The current system runs on single server architecture, which is located in Finland. Here are the main problems areas for the company for doing the up scaling of the current architecture.

In order to keep the response times low the amount of computational power has to be increased or the software needs to be optimised so that it takes less computational power to run. To make the response times caused by the actual transmission of data quicker there are also some options: to make the network distance shorter between the user and the service, to make the connections faster between these two elements or by decreasing the amount of data that needs to be transferred. WordDive's responsibility is

in its own end. The software behind the service should be optimised all the time. Because the way that the customers reach the service can be altered, improvements should also be made on this front. All in all, no matter how optimised the code is, eventually a limit will be reached on how many users one server can handle. Decreasing the response times for the users is the single most important requirement.

Instead of using separate domain names for the different countries, a single domain should be able to be used from anywhere in the world. The main reasons for this are search engine optimisation, social spreading and overall brand recognition.

Another large problem area is the data distribution and its consistency. The data can be divided into separate areas where some of the data is content created by the developers and some of the data is user created e.g. the users' exercise results. For example, these two types of data can be divided so that there is common data that can be e.g. user interface elements and texts. Then there is also data that is different to every user, which is for example their own exercise result. If the application is simply scaled up by using new server farms in new locations, the data in these server farms needs to be synchronised so that the same user can use the same service where ever she is and the data is always consistent. Synchronising the common data, which is mainly created by the developers themselves, might be possible to be handled manually, but when the users create the data it means that the data needs to be synchronised all the time.

When dealing with hardware there is always a possibility that the hardware fails. Therefore, there needs to be a system that works so that if a connection fails to a single server the user is automatically forwarded to use another server. Constant availability of the service is regarded extremely important to keep the paying customers satisfied. Also if the hardware fails, no data should be lost. If the user needs to be forwarded to another server, which probably makes the response times slower, the failed server should be recovered as fast as possible to ensure the best possible user experience.

As it has been for now, the architecture has been designed ad-hoc for the given situation, as is also the case with this new proposed architecture. This means transitioning from the current architecture to the new one needs to be a simple and a planned process and so that the users will not have to deal with long breaks of service. The new architecture should also provide the means for transitioning from that to another architecture.

The new architecture should also be as simple as possible from a non-technical point of view. For example, the content developers' work should not be made more complicated even if the architecture would be geographically distributed. And for the software developer side, the upgrading of the system should stay as simple as possible even if the amount of servers would increase. The best possible case would be such that the geographically distributed nature of the architecture would be abstracted in a way that it would not influence the current workflows of the employees. Also, the architecture should not cause any additional problems for the user, but the user should be able to use the application in the same way as they have used it before. Table 3.1 lists and numbers the requirements mentioned in this section.

Table 3.1 List of requirements for the new system

Requirement Number	Requirement description
1	Low latency to all users around the world
2	WordDive should still be able to be served from a single domain
3	Data consistency and synchronisation
4	High availability and fast recovery after hardware failure
5	Simple transition process to the new architecture
6	Simple structure and usage

3.4 Analysing the case environment

An expert interview was held to strengthen the more theoretical approach to finding the solution. The interview was held with an expert who was then working as the Head of Delivery Capability at Nokia, the Finnish telecommunication company. The interview was held as a semi-structured interview. Some of the themes that were discussed were: the case environment and its special requirements, scalable Web architectures in general, load balancing techniques, cloud services, server distribution techniques and database sharding.

By the expert knowledge in this domain by the interviewed expert and my knowledge of the case environment, we concluded that the amount of events in the case environment is relatively small and most of the data transfer happens from static elements that are identical to all of the users. This means that the application does not necessarily have to be heavy on the database. We also discussed the use of user specific data. Due to the fact that users can access only their own data and not the data of other users, it means that the data of a single user could be held in a single location that would be accessible for that user.

When creating a geographically distributed scalable Web application architecture, a network of servers has to be created. One way to do this is by selecting a 3rd party service provider, who has the international infrastructure ready. This kind of a service might also have software in place, which will handle things such as load balancing and recovering from errors. Another approach is to build the infrastructure by yourself either by negotiating deals with 3rd party Internet service providers to gain access to the servers internationally or by building internal data centres in geographically distributed locations. According to the interviewee, the approach of working with several 3rd party companies might be more troublesome as one might first think, because before one can select a service provider, one should first analyse the provider to find out if it fills the requirements and this can be troublesome when working with companies that do not have a common standard set and when in fact that company is located on the other side

of the globe. Obviously building internal data centres is an expensive solution, but it might still pay for itself since there is no need for a 3rd party and this way you know what you are getting.

When dealing with a start-up Web application like WordDive the planned environment should be flexibly scalable so that money is not spent on anything that is unnecessary. Therefore a solution should be sought that meets the expected growth curve of the application's usage to make the solution as cost efficient as possible. Adding more servers gradually can do this. Some services make this easier and faster than others, e.g. some services provide the customer with the possibility of adding servers by themselves in a matter of seconds. If the addition of servers is done by negotiating new deals with new companies, the process may be significantly slower. Upon selecting the service provider, the service should first be analysed. On the analysis one should analyse things such as the service's availability and network speeds. The network speeds do not necessarily correlate with the geographical distance between the user and the server, but the more accurate measure is the network distance.

Several different solution possibilities were discussed during the interview. For example, the load balancing can be done in many different ways. The first step for making load balancing is by the so-called Anycast methodology. Anycast is discussed in Chapter 2 in detail.

The interviewed expert suggested that selecting a content delivery network might be the best solution for this case environment. This would allow the possibility of keeping the databases and the service logic in the data centre in Finland and moving the static data, which cause most of the network traffic to the new service provider, which has an international infrastructure of servers. Another solution would be to move the whole application to the cloud, e.g. to the Amazon EC2 cloud. But the addition of only the static data to a cloud service like Amazon EC2 might not be a good idea, because the service has too many features for only providing the static content, which means that one might end up paying for features that one might not need.

When one database server meets its limitations, one possibility to scale up is by adding more databases, where one server works as the *Master* server that handles the writing and the other database servers work as the *Slaves* that will only handle the reading of mirrored data. This technique is explained in more detail in Chapter 2. The interviewee also mentioned that it is important to understand how the hardware is used, instead of adding blindly more servers whenever a server meets its limitations.

One recurring subject during the interview was the importance of hiding network latency by the UI. This means that even if the network speed cannot be made faster or it is chosen not to make it faster, it is still possible to make the user feel like it is fast. This happens for example by pre loading elements before the user sees them. Also new techniques should be sought to minimise the amount of database queries, one possible way to do this is by putting the user settings, that are used constantly to some temporary storage.

4 DESIGNING THE NEW ARCHITECTURE

A model of the proposed architecture is presented here. After describing the architecture it is reasoned why this architecture is the best for the given environment. Finally, future architecture options are also presented and compared.

4.1 Comparing different hosting options

It is required that the new architecture is geographically distributed, easily and quickly scalable and affordable. Scaling up locally provides the means for distributing the load between different servers and thus improving the application's performance. This however does not necessarily mean that the application will become faster for the end-user. This kind of scaling will only improve the computational performance, but it will not fix the latency problems caused by the network.

Simply adding more servers to geographically distributed locations around the world by working with hosting companies can at first be an affordable option, however it might take some time to find reliable hosting companies. This will also mean that there will have to be made some customised solutions for how these servers will create a network of computers that will work together, which can make the system more complex and if components such as hardware based load balancers will be bought, this kind of a solution can also become expensive. This kind of a solution could also be made on co-hosted environments, but the same problems could arise. Building the data centres by yourself is an extremely expensive approach at first, but might turn to be more affordable in the long run, however building actually physical data centres is a slow approach.

Cloud-based services provide the flexibility, distributed nature and the affordability that is important for the case environment. Even though cloud based services are affordable at first, at some point they might get more expensive than some other solutions. This means that the use of cloud-based services should be customised as best as possible to the given situation therefore minimising the loss of money. Because it is still uncertain if WordDive's user base will grow to such levels that, for example, building data centres would be a more affordable choice, it can be said that a flexible solution like a cloud-based solution is a more affordable solution for the given situation.

4.2 Combining architectural solutions

In many cases it is not necessary to simply choose one architectural solution, but it is also possible to combine different solutions for different parts of the architecture of the

system. A common first step for a growing Web architecture is to add a content delivery network to transmit the traffic heavy content like images. This will allow to keep the main architecture the same and just to move this content to the content delivery network. This kind of a solution can help considerably with response times if most of the transmitted content is moved into the content delivery network. Due to the geographically distributed nature of content delivery networks, this kind of a solution will improve the experience for all of the users.

4.3 Proposed architecture

Due to the architectural structure of the case environment, it is possible to divide the architecture so that different architectural solutions are used for different parts of the system. Most of the traffic is caused by the transmission of images, sounds and libraries to the client. The problem is not only in the amount bytes that needs to be transferred, but also on the amount of requests. By moving this content away from the current server and into a content delivery network, the load on the current server is decreased and the main focus of the scaling can be done on the content delivery network. A cloud-based content delivery network will further provide the possibility of flexibly scaling up. Figure 4.1 shows what the new architecture could be like when a content delivery network is added to the architecture.

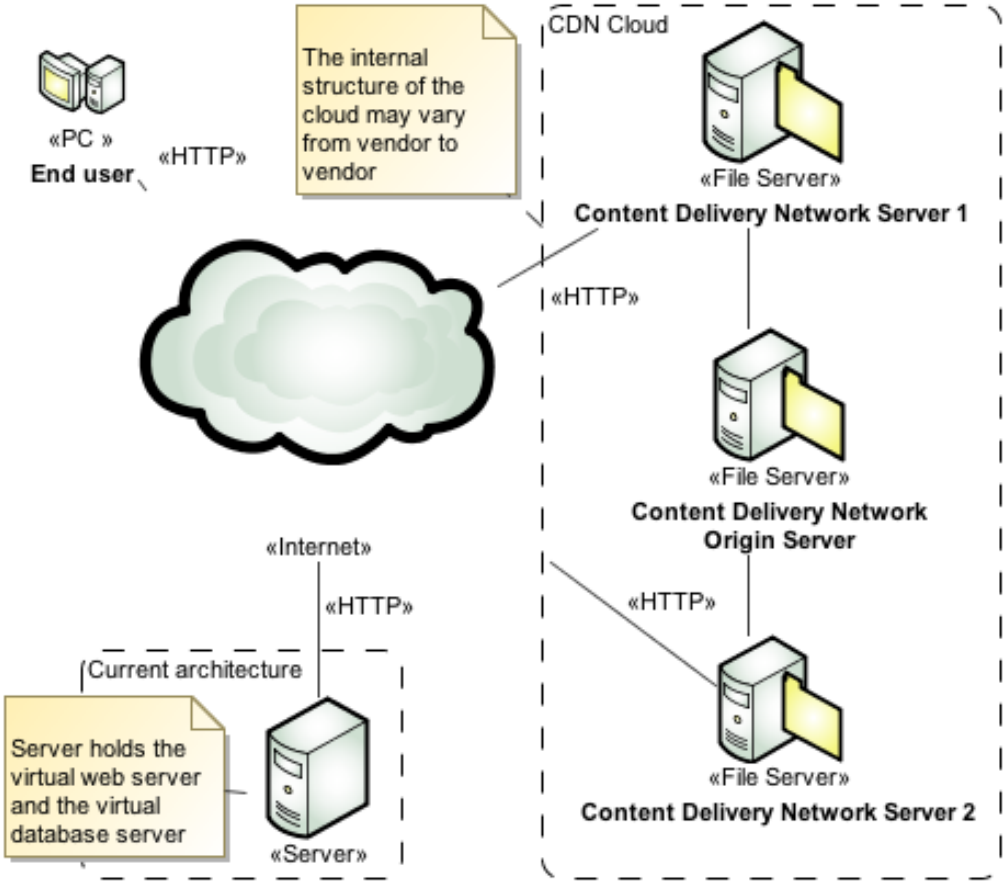

Figure 4.1 Networking diagram of the new proposed architecture

The first step when scaling up to this kind of an architecture is to add the content delivery network. The content delivery network can easily be scaled up by just adding more servers. This might also be done automatically by the service provider. Because content delivery networks are distributed, the location of the new servers should follow the flow of the users. For example, if there is a lot of new users coming in from South America, a new content delivery network server should be added to South America. Thus reducing latency for these users. If a single content delivery network does not meet the requirements, another content delivery network from another service provider can also be added to the system.

Adding just a content delivery network can be seen as a simple addition to the architecture, which might require only minor changes to the application code. The case environment's main problem is the latency problem with the customers from faraway. A content delivery network fixes just that. Depending on the service provider a content delivery network should be a more affordable option than actually setting up Web servers all around the world, because a content delivery network provides a narrower feature set.

The downsides for this kind of a solution come when the problem is more complex than simply a latency problem. Addition of a content delivery network will not help e.g. with database performance problems. The price of the service can consist of things such as the amount of requests made to the server or the amount of gigabytes that have been transferred. Therefore it should be kept in mind that if for example the price of a single request is expensive, then it might not be affordable to put e.g. 1-kilobyte images on the content delivery network, as it might be more affordable to serve that image in some other way.

4.4 Evolution of the architecture

The first step of scaling up is just to add more servers to the content delivery network. This should take care of most of the latency issues, as the content delivery network will handle most of the traffic. However when the system's backend servers start to reach the end of their capabilities, the architecture will need to be further scaled-up. Two different ways of scaling-up are presented here. The first option is a more traditional way of adding more servers and load balancers. The second option is to move the original architecture into a cloud-based service.

4.4.1 Option 1

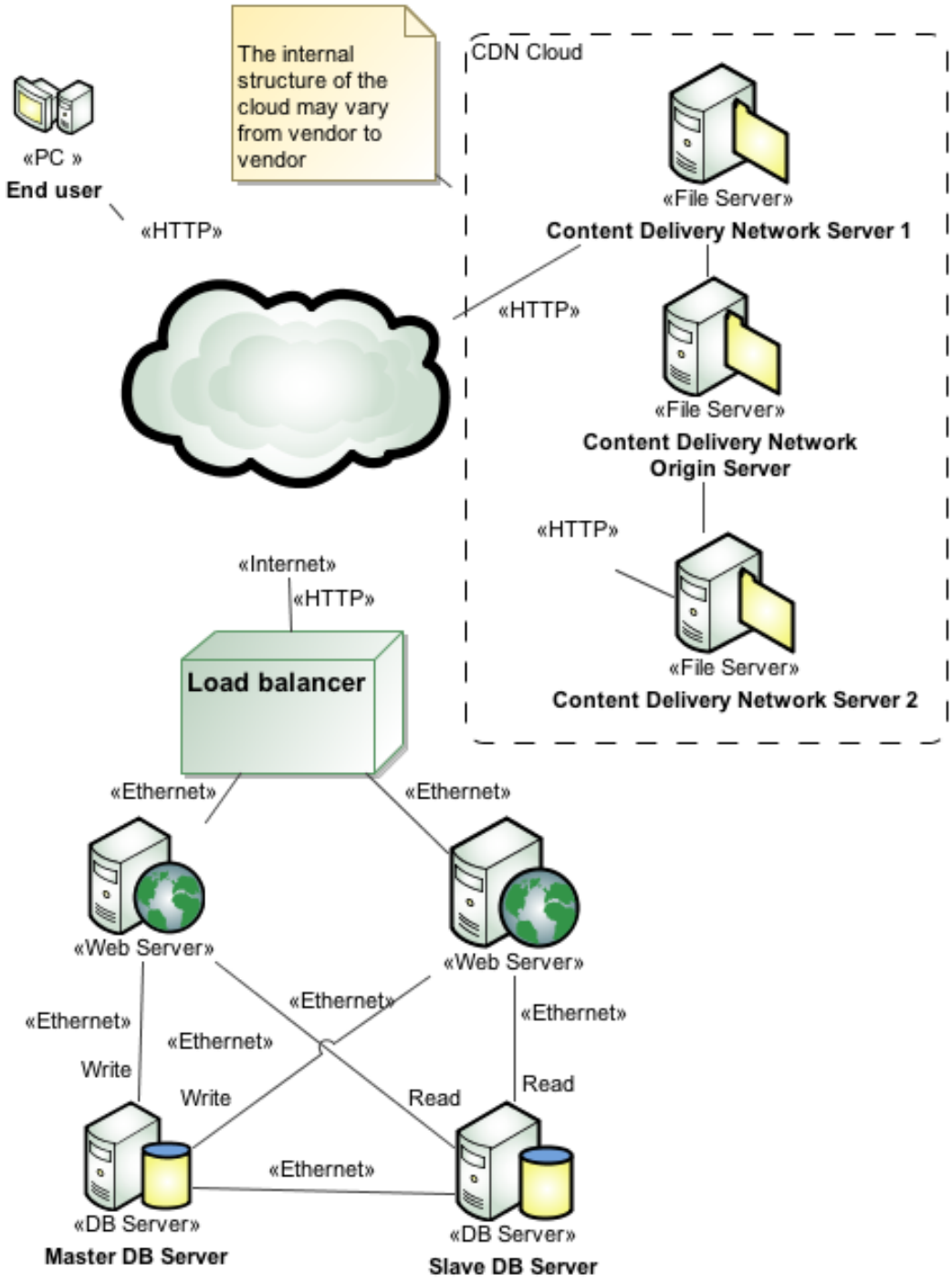

Figure 4.2 Evolution of the proposed architecture, option 1

As shown in Figure 4.2, the content delivery network is kept untouched, but the original one server system is changed into a cluster of servers. To scale more flexibly, this cluster can be made one server a time. This should not be done too hastily as the addition of servers can get expensive especially if hardware based load balancers are going to have to be used. Before adding these servers the cause of the load should be

determined, that is to say that there is nothing to be gained from adding a new Web server if the problem is in the database's read performance.

To horizontally scale the Web servers, a load balancer of some sort needs to be added that will handle the division of traffic. The other Web servers are mere replicas of the first server. If any updates are made to the Web server, the updates will have to be made to all of the Web servers. The database servers can also be horizontally scaled by adding *Slave* servers. The *Slave* servers will handle the read operations to the database. So this kind of a solution will only increase the read performance. However if the *Master* server cannot handle the amount of write operations, other solutions will have to be sought. One possible solution is to shard the database on the database engine's level so that no major changes will have to be done to the application logic. These subjects are discussed in Chapter 2 in detail.

4.4.2 Option 2

Scaling Web and database servers on a traditional Web hosting service is not as flexible as in a cloud-based service. Instead of just adding a content delivery network to the cloud, the whole system can be moved into the cloud. This will be a total overhaul of the current architecture, as it needs to be fitted into a new environment. However in the end, this kind of an architecture will be quite simple. The basic idea is shown in Figure 4.3.

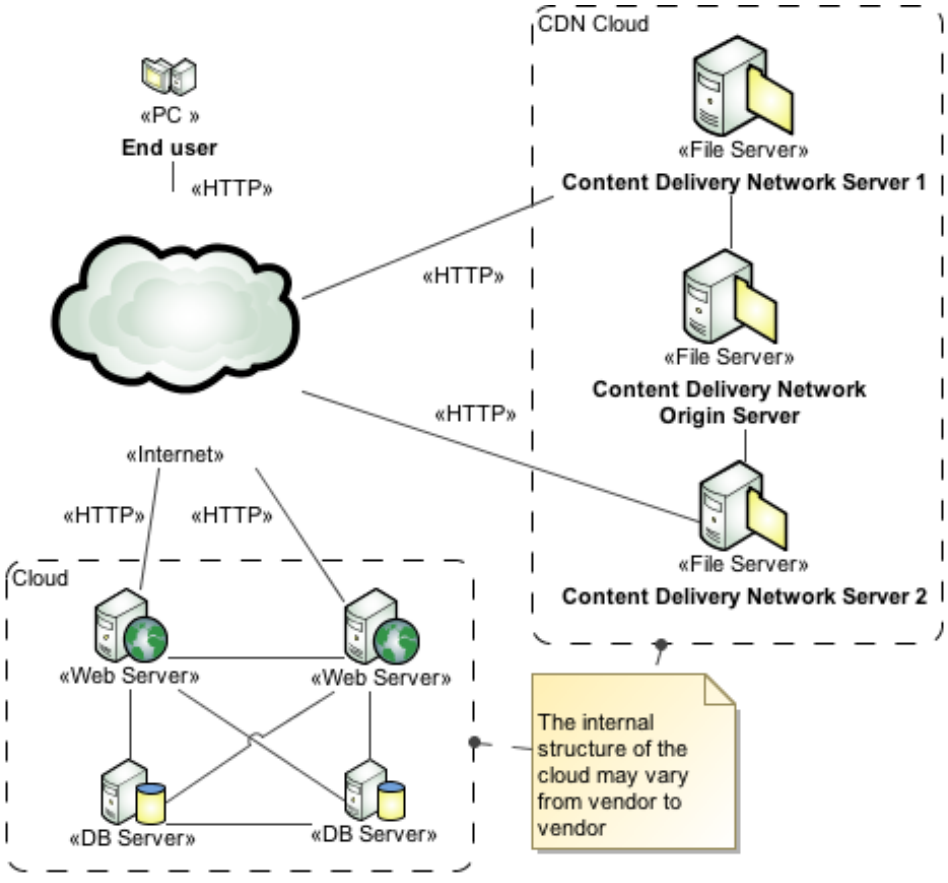

Figure 4.3 Evolution of the proposed architecture, option 2

This kind of a system follows the idea of *infrastructure as a service*. This means that the customer does not have to care about the hardware at all. The cloud service provider takes care of all of the hardware issues and the customer can focus on managing the features that are needed. Scaling up or down this kind of a system is made extremely easy and as such is extremely flexible. This makes it possible to always strive to use the ideal amount of servers to serve the website.

Moving all of your data to the cloud means that you will also have to trust all your data to the cloud service provider. Moving away from a Web hosting company with a trusted partnership might seem risky to some people. Special care should be taken while reading the cloud service provider's *service level agreement* (SLA). Moving from a local Web hosting service to a global cloud service provider may raise questions like, what to do if for example the database server stops responding. Can the service provider be reached over phone or is there any kind of customer service at all. The price of this kind of a service may consist of many different pricing schemes, such as hourly based pricing, gigabyte based pricing or request based pricing. Therefore the actual price of such a service may be difficult to calculate.

4.4.3 Comparison of the options

The future development of the architecture can be done either way. The best and most affordable way has to be determined when the time comes for it by reviewing the different possibilities. Option 1 offers a traditional way of scaling up, by adding more servers to the current system. This is a safe and logical solution, but it may get too expensive due to elements such as hardware based load balancers. The other problem is that this kind of a solution can only scale to some point until the Web hosting company has to be changed. WordDive's current Web hosting service provider is a Finnish Web hosting company, which does not currently have the same kind of capabilities as some of the global cloud service providers.

Scaling the architecture up with option 2 has the upside of more flexibility. Instead of adding one server at a time to the system, a cloud-based system can be constantly managed so that there is never too many or too few servers up and running. This kind of a system can also scale up with less hassle, since the scaling up happens only by adding more computing units to the system, literally by the click of a button. The customer company does not have to care about things such as the addition of load balancers. Then again this abstraction of the hardware can make the developer uncertain of what actually needs to be scaled up. When dealing with actual hardware, you have the possibility to gather all the data that you need to make the decisions. Cloud-based platforms may have some requirements for the environment that is going to be added to the cloud, which might require application level changes. For example, a relational database might create problems with some cloud service providers.

The final decision has to be made when the time comes for it. All in all it is a numbers game. Cloud-based services have been developing fast during the last years

and no doubt will continue to develop. The idea of having the servers running on huge purpose built data centres in the most optimised way, promises less costs with good quality. In the end when the time comes for it, the different possibilities and their prices need to be calculated and the decision will have to be made.

4.4.4 Further possibilities

Scaling-up does not have to end to previously mentioned design decisions. Optimising the code, the database and the content can severely help to decrease the load on the servers and to decrease the amount of bandwidth that is needed. One common solution is also to add caching to the system. When using content delivery networks, caching is probably already in use. However caching mechanisms can also be added to work with the backend servers, so that for example instead of making the read operations from the database, they can be read from the cache.

5 TESTING

Tests were devised to see if the architecture works in practice in the way that it theoretically should. In this chapter, first the test environment and the test plan is described, then the results are presented and finally the results are analysed.

5.1 Approach

As discussed in Chapter 4, a content delivery network is added to WordDive's current architecture to improve the latency for users living in countries faraway from Finland. This solution is tested by selecting a single service provider as a representative of this kind of solution. Simple test applications are created to imitate the day-to-day usage of WordDive. These applications will be run from several locations around the world so that these applications are running on the current WordDive server without the content delivery network and so that the content delivery network is used.

5.2 Test applications

The testing will consist of using two different web pages that are being loaded. The first test application opens up the free trial exercise. Opening up the free trial forces the user to download the Web application, which is used for running the exercise in WordDive. This includes all of the libraries related to it and also some other commonly used files. In addition to this, opening up the free trial requires that the web page is created dynamically and requests are made into the database. A small latency in this feature is a definite must to ensure a proper user experience, because this is the first time that the user is using WordDive. The first test application is a modified version of WordDive's free trial feature. The modifications are done to remove any sources of interference caused, for example, by requests sent to other Web services. This test application's content is mainly made up of image and JavaScript files. While there are 23 images, still the 10 JavaScript files build up most of the bytes that need to be transferred.

The second test application will test the content delivery network's ability to improve latency on the actual WordDive exercise. This test application will be a simple Web page with different images from the WordDive exercise. This is tested, because it is the single most used feature in WordDive and it is used by new users and also by existing WordDive customers. This test application is made up of 10 image files and a HTML file. While in a real situation the images would be random, in this test case the images are kept the same, so that in each test the amount of bytes downloaded will stay the same.

5.3 Choosing the service provider

Content delivery networks have been around for several years already; this has allowed many companies to build businesses that provide content delivery network services. The biggest and longest running content delivery networks are mostly provided by companies that provide premium services that do not have listed prices on their website, but the price is negotiated with a sales representative. The price may vary depending on the customer.

One of the biggest cloud service providers at the moment is Amazon with its Amazon Web Services cloud computing platform. Because this thesis's purpose is not to do market research of the different content delivery network service providers, Amazon Web Services is chosen as the platform for the simulation due to its good reputation especially among start-up companies. Amazon Web Services can be regarded as a de facto standard choice among Finnish start-up Web companies.

Part of the Amazon Web Services is the Amazon CloudFront service, which is a cloud-based content delivery network. Amazon CloudFront's website (2011) describes how the service operates on a pay-as-you-go basis, which means that the pricing consists of the amount of data being transferred out of the network, the amount of requests being sent to the network and the amount of invalidation requests that have been made. CloudFront content delivery network is currently distributed between three different continents. The network consists of ten different locations in North America, six different locations in Europe and three different locations in Asia.

CloudFront is used through APIs and the Amazon Web Services Management Console, which is a simple graphical user interface tool for managing Amazon Web Services. The files are added to the network through an origin server. This origin server can be part of the Amazon Web Services platform or an external server. The files are moved to the cloud by calling an API provided by the CloudFront service. Finally, the links to these files are added by changing the existing application code using the new domain names provided by the content delivery network. Amazon CloudFront determines the best location to serve the content when a request is made. The content can be delivered using HTTP or HTTPS protocol. It is also possible to restrict the access to the files through a private content feature. This feature makes it possible to let only certain signed clients to reach the private content. The invalidation of files is done by setting an expiration period on the files and by using the Invalidation API. Only the most frequently used content is cached on the edge servers, which means that the performance is dependant on the popularity of the given file. If a file is not popular enough, it is served from the origin server, which can severely increase the latency. A file may also be served as a stale file, which means that the edge server will first check from the origin server if there is a newer version available and only then it will serve the file. Amazon CloudFront does not currently automatically compress the files that it is serving. (Amazon. 2011.)

5.4 Test environment

The test environment consists of two major parts. First of all, a development environment is set up on WordDive's current Web server, which has the test applications running on it. Often the most reliable way to measure load times is by using real user data. In this case it is not done, due to the lack of substantial enough user base in faraway countries and because the testing is wanted to be done in a tightly controlled environment. Running the tests on a development environment is a safe option, because it will not risk the user experience of the users that are actually using WordDive at the time of the tests. The development environment is an identical copy of the production environment running on the same piece of hardware. This means that the behaviour of the WordDive users can affect the tests. If there are a lot of people using the service at the time of the tests, the server and the network will have more load to handle. This is why the tests on the different test applications are run as close to each other as possible and some of the tests are repeated on different times of the day.

The second part of the test environment consists of the computers that send the requests for the Web server and measure the time it takes to load the Web pages. This part of test environment is provided by the Internet monitoring company Keynote Systems. The software application in particular used for testing is the Web version of KITE (Keynote Internet Testing Environment), which is a free version of Keynote's service[1]. One part of the service is the ability to test load times simultaneously in 6 different locations around the world. These cities are San Francisco, New York, London, Frankfurt, Hong Kong, and another measuring point in San Francisco, which is connected to a DSL broadband connection. These cities provide a good overall picture of the Web application's performance in locations around the world. The service provides numerical data and graphical data after each test run has been executed. The results are collected and in the end all of these results are combined. The Keynote's service is trusted to be reliable enough so that it is not influenced by factors such as the load on the test server itself or by caches on the Web browsers or in the network. The test results are analysed and special care is made to find any fluctuations in the data that might be a sign of such error. The testing is done to find out the perceived response time for real world users and more specifically how it improves. Keynote's service provides the ability to run the tests on the Firefox or on the Internet Explorer browser. This means that the tests are actually run on real Web browsers, while running the tests on simulated browsers might show results that are inconsistent with the real life scenario.

5.5 Test preparations

The preparations start from setting up an origin server, which serves the content to the content delivery network. Amazon S3 permanent data storage service is used for this, because it requires fewer configurations and because it is cost efficient. The files that

[1] KITE can be accessed through http://kite.keynote.com/

are added to the origin are found out by studying the test applications and by figuring out, which files they use. Once the origin server is set up, it is time to populate the content delivery network. This is done through Amazon Web Services Management Console. A content delivery network distribution is created and the origin server is simply selected from the GUI. After a while the data from the origin server is distributed throughout the global content delivery network. The data in the content delivery network can then be reached from the domain provided by Amazon.

The test applications are served from a test environment, which is running on WordDive's current server. The test environment is almost identical to the production version. The test applications are added to this test environment, where they are accessed from four different URLs. Two of the test applications are using the new domain provided by Amazon as the source of these files.

At this point the test environment is fully set-up and the tests can be run. The tests are run from http://my.keynote.com/ from the Test tab. The URL of the test application is added, the Internet Explorer browser and all of the test cities are selected and the test is run. Internet Explorer is chosen as the browser of choice, because Internet Explorer is the most used browser among WordDive users. The testing tool is shown in Figure 5.1.

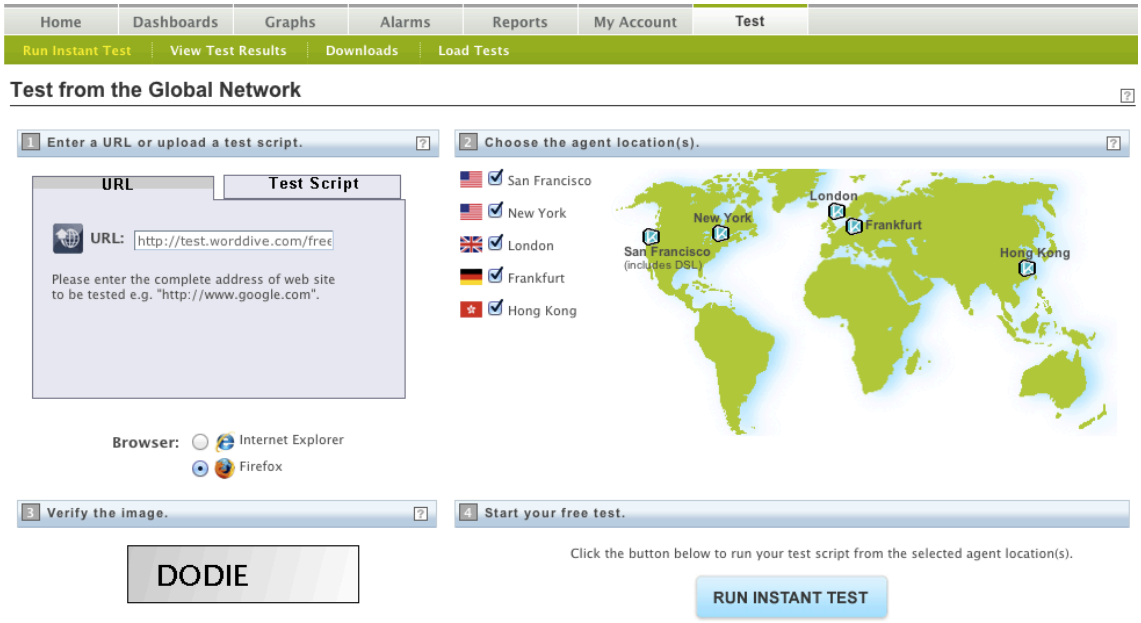

Figure 5.1 Screenshot of the Keynote testing tool

The test basically completely loads the given webpage and measures the time it takes. The results of the test are copied from the testing tool's results view. The provided data include the User Experience and Network times in seconds, the amount of errors and the amount of bytes that where downloaded. It is also possible to see more detailed information. Figure 5.2 shows the testing tool's results view.

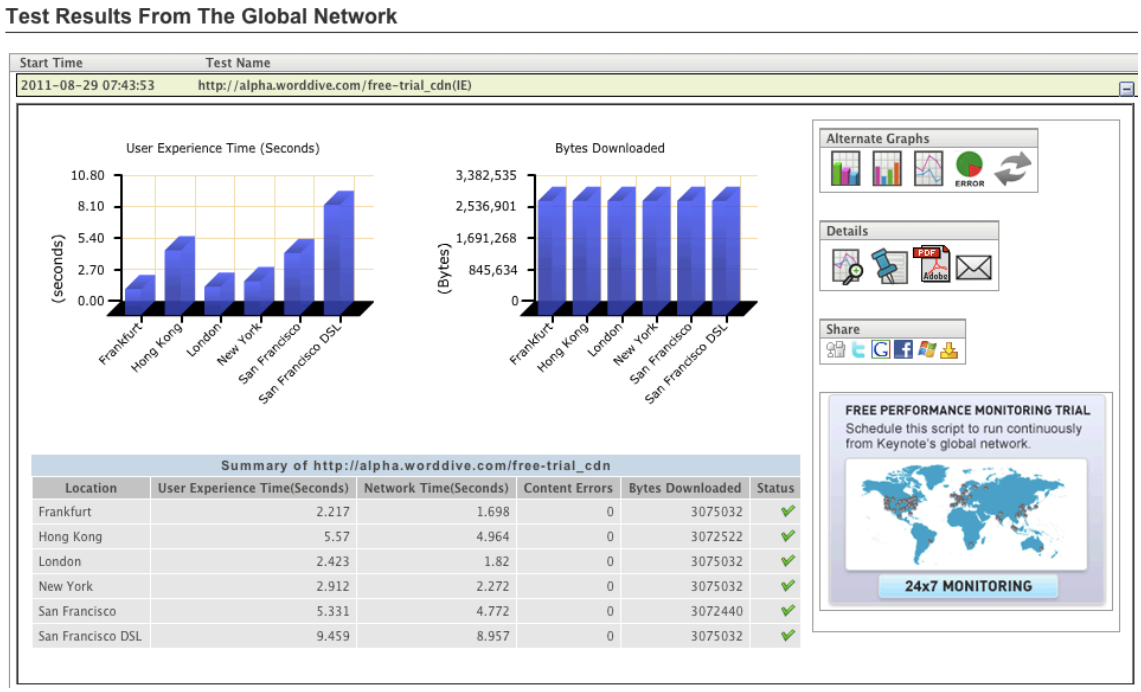

Figure 5.2 Screenshot of the Keynote testing tool's results view

5.6 Test plan

There were two different test suites planned. Because the content delivery network is essentially a set of network caches, it is expected that there may raise some differences depending on the length of the break between the tests. Therefore the first test suite will have breaks of more than one hour between the tests. This will imitate the behaviour if there are not many users using the Web application and so the content delivery network will not necessarily see the files as popular. The tests will be repeated 15 times on each of the test applications on different times of the day in a period of four days. This is done for two reasons. Firstly, adding time between the tests will allow the caches some time to clear. Secondly, if there is too much load on the WordDive server or on the CloudFront servers on some particular time of the day, several measurements are done on different times so that this error will not show up on all of the measured results.

The second test suite will do the same tests with minimal breaks between the tests. Tests will be run every five minutes. This will be repeated 15 times. The order at which the test applications are run will also be changed to be opposite than in the first test suite. This will make it possible to analyse if external Web caches influence the results depending on the order at which the test applications are ran. This test suite will measure the content delivery network's performance in the case where the use of the test applications is frequent.

5.7 Test results

The results are divided into sections 5.7.1 and 5.7.2, showing calculated values from all of the test applications in each of the test locations. In addition, all of the test results are

shown as diagrams from each of the test locations. Finally, these results are compared in Section 5.7.3. Overall there were 720 measurements done in six different locations and by four different test applications. This section will show these results by the use of diagrams and tables. All of the measurements use the network time measurement instead of the user experience time. This is done because the user experience time can be influenced by the performance of the computer that the tests are run on.

5.7.1 Test suite 1: Non-frequent usage

Table 5.1 shows results from test suite 1. The values are calculated from the values that are presented in Figures 5.3-5.8.

Table 5.1 Combined test suite 1 results

Test 1	Mean	Standard Deviation	Median	Minimum	Maximum
Frankfurt	3.68	0.04	3.685	3.592	3.744
Hong Kong	25.82	0.97	25.571	25.064	29.09
London	4.52	0.1	4.506	4.359	4.702
New York	10.59	0.16	10.566	10.282	10.89
San Francisco	17.81	1.07	17.457	16.938	20.337
San Francisco DSL	18.66	0.9	18.273	17.918	21.312
Test 1 CDN					
Frankfurt	1.87	0.8	1.616	1.22	4.425
Hong Kong	16.68	7.71	19.904	4.774	30.13
London	2.16	0.45	2.034	1.67	2.999
New York	3.06	1.08	2.489	2.05	5.087
San Francisco	6.11	3.24	4.521	3.714	16.17
San Francisco DSL	9.48	1.63	9.158	7.834	13.704
Test 2					
Frankfurt	0.62	0.01	0.618	0.613	0.663
Hong Kong	4.67	0.11	4.692	4.543	4.943
London	0.76	0.02	0.755	0.728	0.802
New York	1.9	0.03	1.901	1.837	1.952
San Francisco	3.19	0.1	3.178	3.082	3.501
San Francisco DSL	4.81	4.87	3.452	3.258	22.369
Test 2 CDN					
Frankfurt	0.33	0.12	0.296	0.187	0.571
Hong Kong	3.99	2.26	3.618	0.77	9.117
London	0.41	0.14	0.36	0.25	0.707
New York	0.59	0.37	0.405	0.281	1.368
San Francisco	1.55	1.21	0.746	0.582	4.38
San Francisco DSL	1.65	0.85	1.249	0.861	3.545

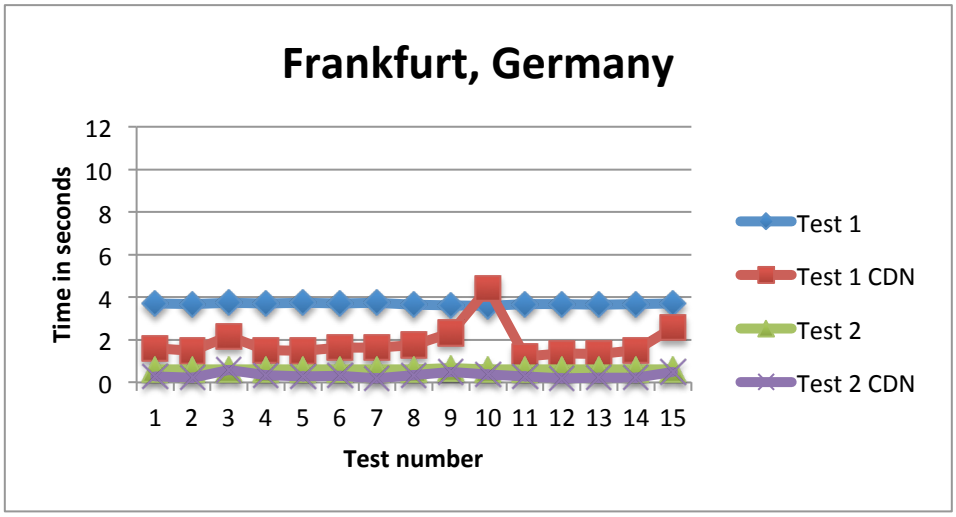

Figure 5.3 Test suite 1 results from Frankfurt, Germany

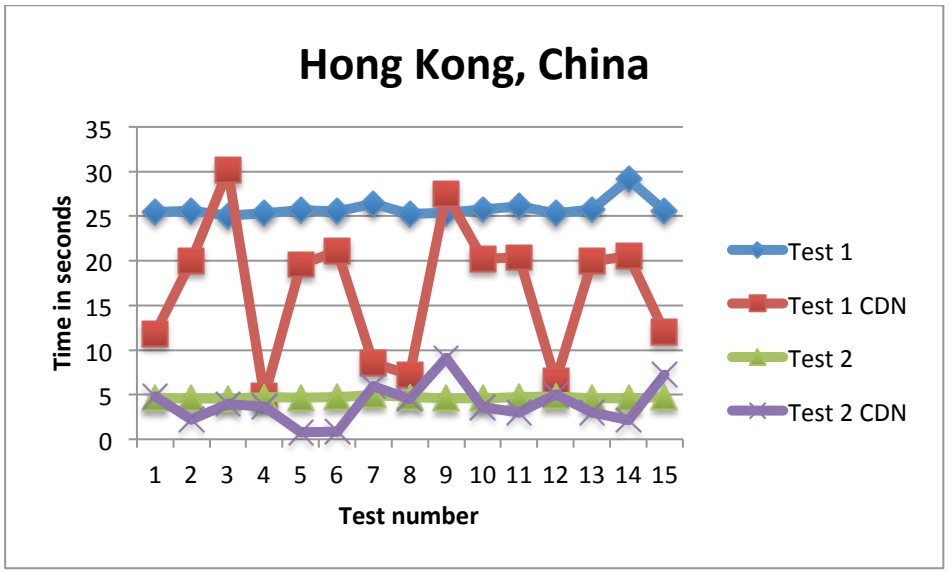

Figure 5.4 Test suite 1 results from Hong Kong, China

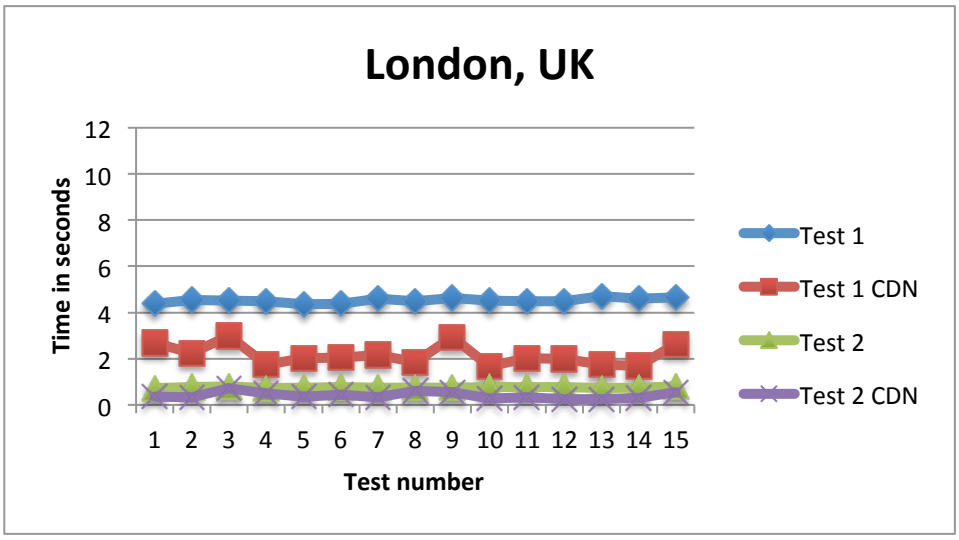

Figure 5.5 Test suite 1 results from London, United Kingdom

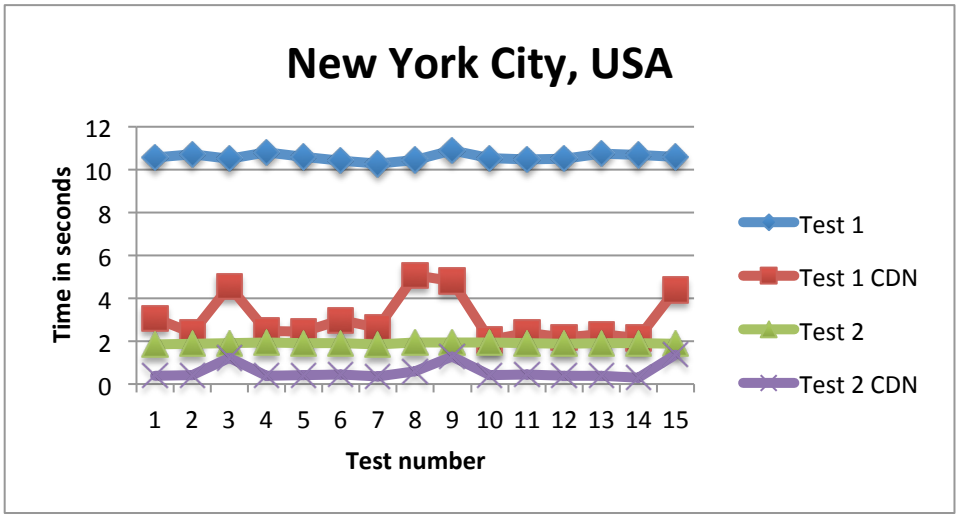

Figure 5.6 Test suite 1 results from New York City, United States of America

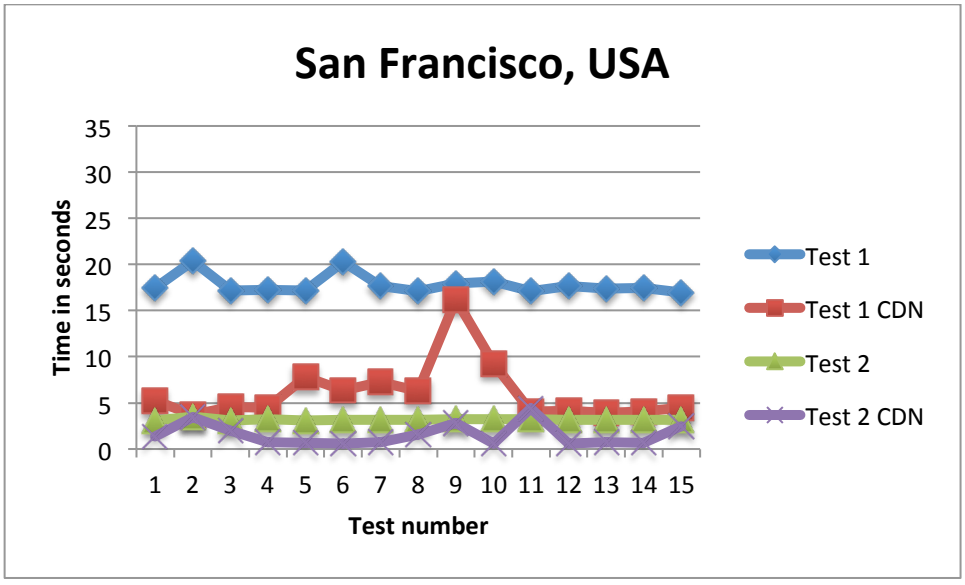

Figure 5.7 Test suite 1 results from San Francisco, United States of America

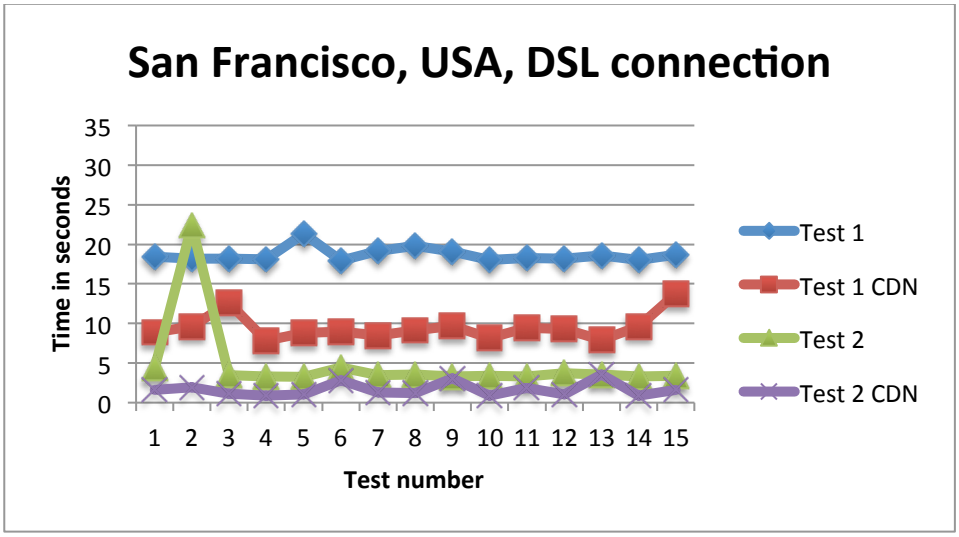

Figure 5.8 Test suite 1 results from San Francisco, United States of America, DSL connection

5.7.2 Test suite 2: Frequent usage

Table 5.2 shows results from test suite 2. The values are calculated from the values that are presented in Figures 5.9-5.14.

Table 5.2 Combined test suite 2 results

Test 1 CDN	Average	Standard Deviation	Median	Minimum	Maximum
Frankfurt	1.34	0.35	1.165	1.021	2.206
Hong Kong	13.92	9.52	9.305	4.092	33.13
London	1.78	0.14	1.785	1.614	2.164
New York	2.22	0.19	2.144	2.006	2.629
San Francisco	4.24	0.44	4.128	3.671	5.184
San Francisco DSL	8.58	0.71	8.308	7.786	10.493
Test 1					
Frankfurt	3.97	0.91	3.67	3.599	7.092
Hong Kong	25.5	0.28	25.495	25.055	25.97
London	4.42	0.07	4.402	4.309	4.513
New York	10.59	0.13	10.614	10.377	10.78
San Francisco	17.84	1.19	17.397	16.998	21.103
San Francisco DSL	18.09	0.35	18.029	17.639	18.965
Test 2 CDN					
Frankfurt	0.2	0.11	0.148	0.13	0.551
Hong Kong	2.92	2.11	2.881	0.668	6.563
London	0.32	0.16	0.256	0.234	0.831
New York	0.44	0.41	0.346	0.284	1.908
San Francisco	0.86	0.72	0.663	0.52	3.267
San Francisco DSL	1.12	0.56	0.952	0.84	3.101
Test 2					
Frankfurt	0.62	0.01	0.615	0.612	0.643
Hong Kong	4.67	0.1	4.614	4.567	4.905
London	0.77	0.02	0.759	0.734	0.801
New York	1.91	0.04	1.916	1.85	1.964
San Francisco	3.16	0.06	3.158	3.006	3.273
San Francisco DSL	3.74	0.59	3.342	3.242	4.939

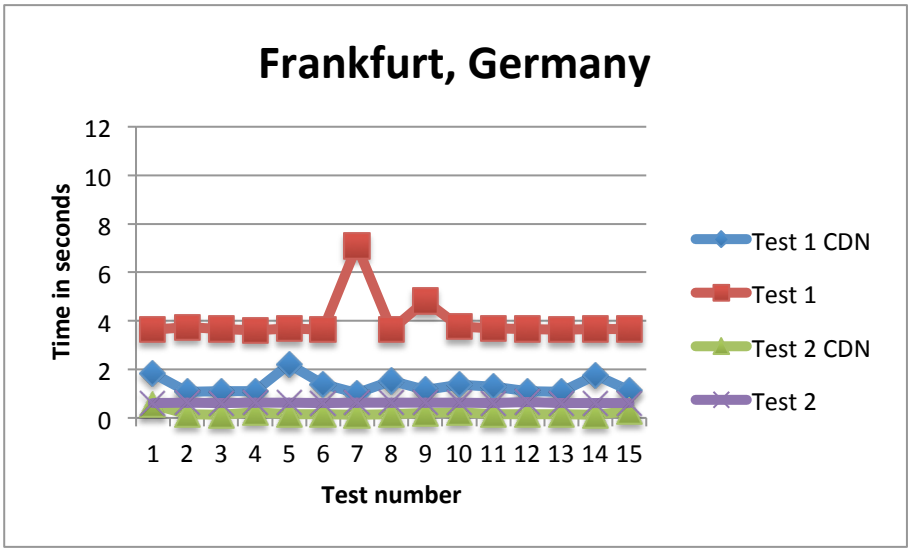

Figure 5.9 Test suite 2 results from Frankfurt, Germany

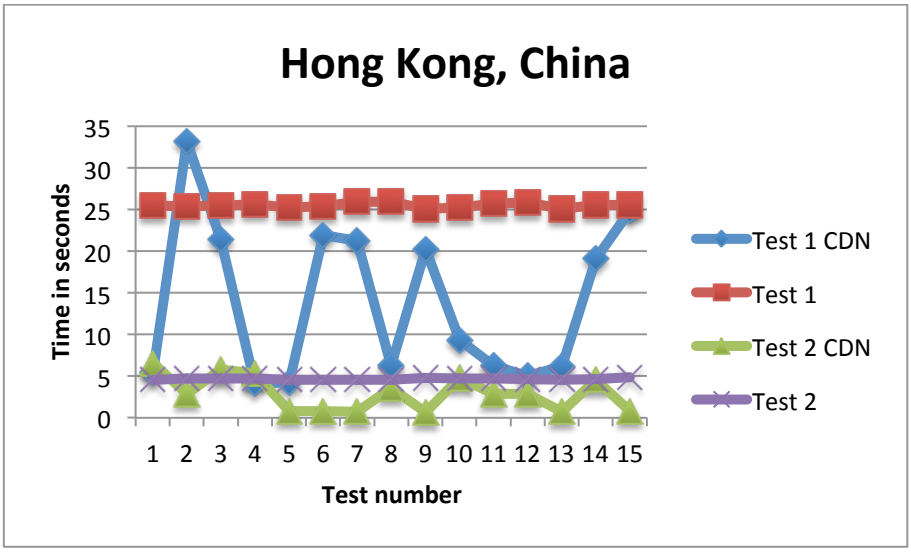

Figure 5.10 Test suite 2 results from Hong Kong, China

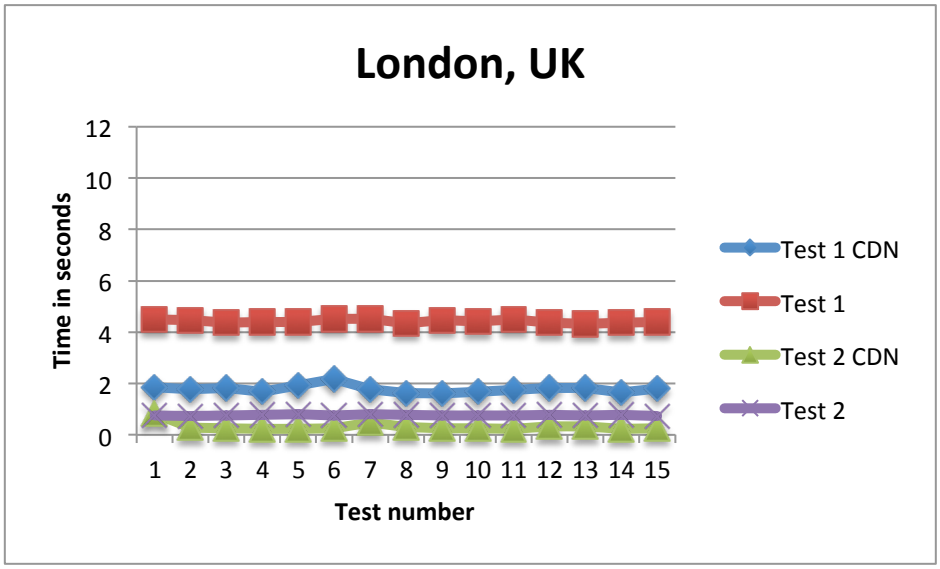

Figure 5.11 Test suite 2 results from London, United Kingdom

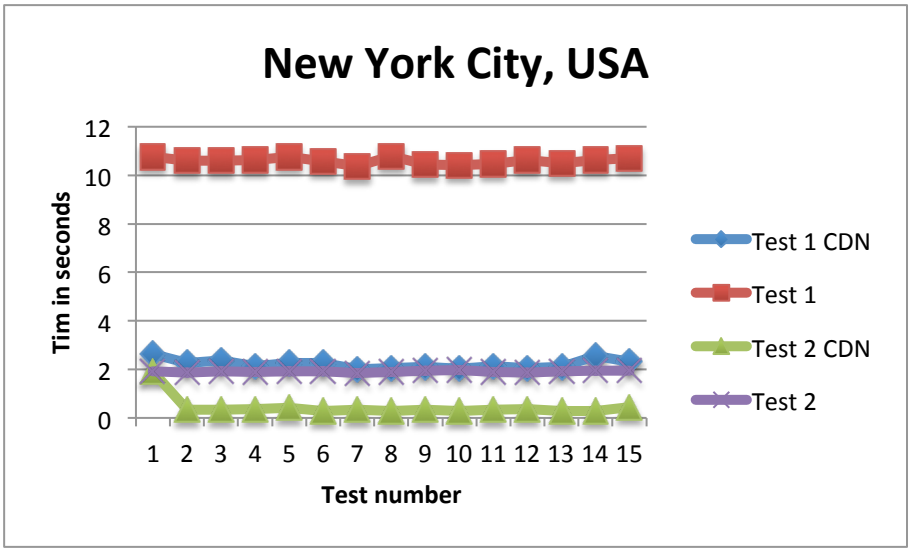

Figure 5.12 Test suite 2 results from New York City, United States of America

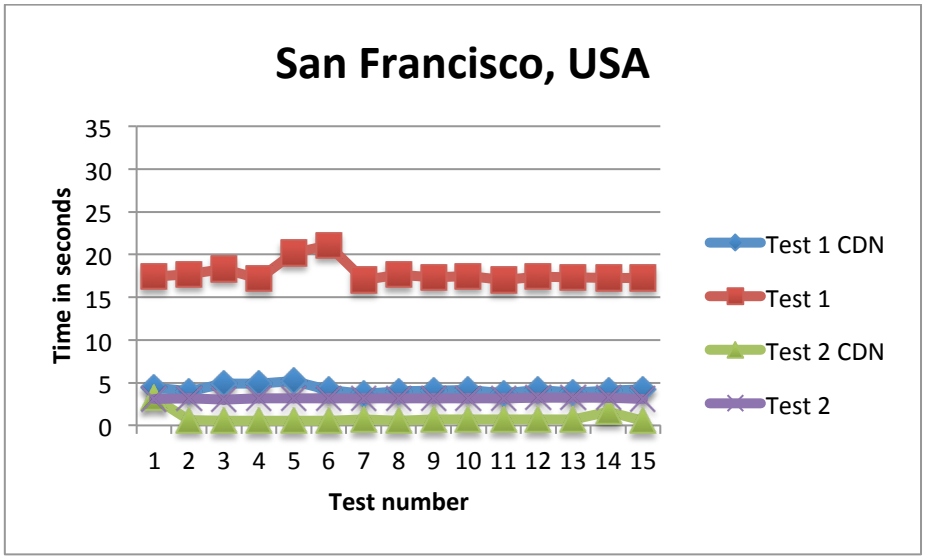

Figure 5.13 Test suite 2 results from San Francisco, United States of America

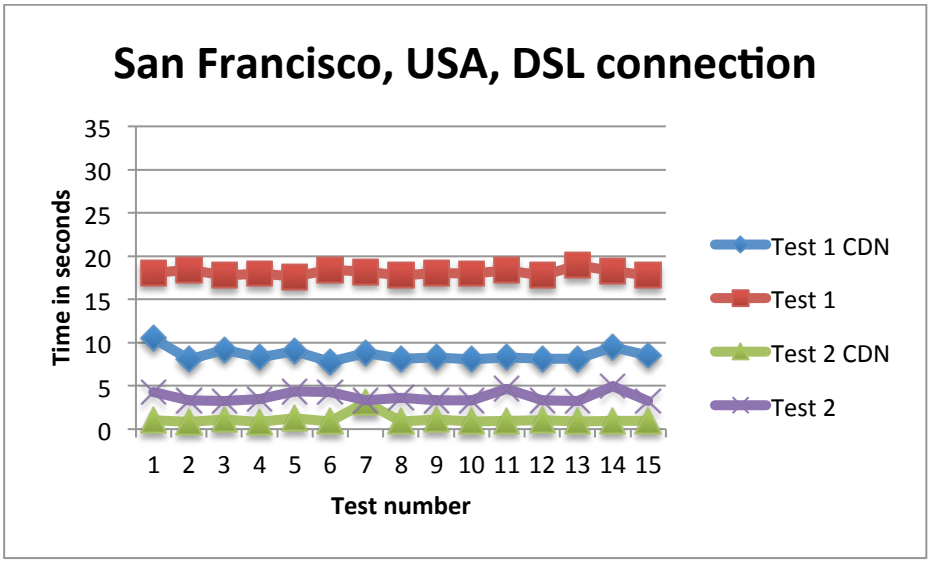

Figure 5.14 Test suite 2 results from San Francisco, United States of America, DSL connection

5.7.3 Combined results from the test suites

Table 5.3 lists the amount of bytes that were downloaded with each of the test applications. Table 5.4 shows a comparison of each of test suite in each location. The comparison is done between the two test applications, one with the content delivery network and one without it. The percentages show the improvement made by the addition of a CDN.

Table 5.3 The amount of downloaded bytes with each test application

Test Application	Bytes downloaded
Test 1	959070
Test 1 CDN	3074872
Test 2	224394
Test 2 CDN	228948

Table 5.4 The comparison of the improvement percentage of the response times between CDN and non-CDN test applications

Test application 1	Test Suite 1, Mean	Test Suite 1, Median	Test Suite 2, Mean	Test Suite 2, Median
Frankfurt	196.79 %	228.03 %	296.27 %	315.02 %
Hong Kong	154.8 %	128.47 %	183.19 %	273.99 %
London	209.26 %	221.53 %	248.31 %	246.61 %
New York	346.08 %	424.51 %	477.03 %	495.06 %
San Francisco	291.49 %	386.13 %	420.75 %	421.44 %
San Francisco DSL	196.84 %	199.53 %	210.84 %	217.01 %
Test application 2				
Frankfurt	187.88 %	208.78 %	310 %	415.54 %
Hong Kong	117.04 %	129.68 %	159.93 %	160.15 %
London	185.37 %	209.72 %	240.63 %	296.48 %
New York	322.03 %	469.38 %	434.09 %	553.76 %
San Francisco	205.81 %	426.01 %	367.44 %	476.32 %
San Francisco DSL	291.52 %	276.38 %	333.93 %	351.05 %

5.8 Analysis of the results

Tables 5.1 and 5.2 show the combined results from the tests. Due to Web performance measurement data's complex nature, there are several different ways to ways to analyse the data. As can be seen from Figures 5.3-5.14, the measured data is not normally distributed, but it is actually positively skewed. This can be seen by the great number of spikes on the lines and by the lack of drops. If you were to draw a straight line, where the central tendency of the values lies, there would be more points above the line than below it. Meaning that the data has the tendency of having some pieces of data that are in the upper end of the set. Because the mean value is impacted so much from these values, the mean value can actually be far from the central tendency. As Jain (2006)

suggests, with this kind of skewed data, it is better to use the median value than the mean value.

Much can also be seen from the standard deviation values. The tests that were done without the content delivery network have lower standard deviation values. The tests that were done without the content delivery network gave fairly uniform results, which means that the testing environment and the testing tool were stable and that external elements like an ISP's Web cache, did not noticeably influence the results. The tests that were done with the CDN in the first test suite failed to give stable results and there was a lot of variance in the results. As the test configuration mimics the real life situation, it can be said that while there is a lot of variance in the results, it is also true that this kind of variance can happen in a real production environment. This variance was inspected. The suspected reason for such large variance especially with the Hong Kong results can be found from the behaviour of the CloudFront content delivery network. The content delivery network only keeps the most popular files on the edge servers. The files were probably dropped from the CloudFront's Asian servers' cache, which made the variance to tests. Test suite 2 did the tests with shorter breaks. The response times and the standard deviation decreased. Even though the Hong Kong tests showed improved performance, there was still large variance in the results. This shows that the CloudFront content delivery network fails to give the same consistent performance boost in Asia than what it does in North America.

Minimum and maximum values point out the best and worst case scenarios of each test application. In most cases the tests with the content delivery network provided a much smaller minimum response time and also a much smaller maximum response time. However, due to the variances in the test results, there were also some results where the maximum value was larger than without the content delivery network. Decreasing the length of break between the tests in test suite 2 did not noticeably change the minimum and maximum times. The main difference between the test suites can be seen from the average and median values in the tests that used the content delivery network. Tests without the content delivery network did not show noticeable differences, which also means that the running order of the test applications did not have notable impact on the tests.

Figures 5.3-5.14 present the data distribution of the test results in different cities and in the different test suites. The non-CDN tests on all of the figures show fairly uniform data distribution, although some of the results are slightly skewed by tests that had a much larger response time than the rest of the tests, this can be seen, for example, in Figure 5.8, where the line labelled "Test 2", shows a large spike. Test suite 1 results with the CDN show results with a lot of variance. Test suite 2 results with the CDN however show a trend, where the first measurement has a higher value than the rest of the values. This shows how the content delivery caches the files and after the first try, they are quickly served directly from the edge server's cache. The results from Hong Kong with the CDN show poor consistency in both of the test suites. Nonetheless, it does seem like around half of the tests show results that are around the 5-second area

and the other half on the 20-second area. This is most probably a sign of the content delivery network failing to keep the files on the edge server. This leads to the files being downloaded from the origin server, which was in this test environment located in Dublin, Ireland.

Table 5.3 shows the differences between the sizes of the test applications. The test applications themselves use the exactly same files and therefore they have the same sizes. The size differences are created by the manner at which the files are transmitted. WordDive's Web server is able to compress the files automatically, when it recognises that the Web browser is able to uncompress the files. This is extremely useful especially with the code libraries that compress well. As can be seen from the Test 2 results in Table 5.3, the size difference is not that big, because the images that where transmitted were already compressed and compressing them again, did not make much difference. The CloudFront content delivery network is not able, by default, to compress the files. However, there exists a workaround for this. Compressing the files in the content delivery network would probably further improve the response times. Be that as it may, the size improvements would not be directly proportionate to the percentage that the response times would decrease. Even though the file sizes influence the response times, there are also a lot of other things that influence them.

As the new architecture's main purpose was to improve the response time for users in faraway countries, the results are compared with the old and the new architecture. As shown in Table 5.4 the improvement ranges from 117% to 553%. These numbers show the worst and the best-case scenarios. If the content delivery network is functioning as well as it should, it can be seen that the response times decrease to a half or even to one fifth of what the response times were. As pointed out earlier, the numbers based on the median are more reliable. The best results where found from the tests ran in North America. There are at least two reasons for this. Firstly, WordDive is currently served from Finland. There is less distance to the European locations than to North America from Finland, so the performance was already quite good in the European locations. Secondly, CloudFront has more servers in North America than in any other continent, which is probably the reason, why it has better performance in there than e.g. in Asia.

5.9 Test conclusions

The tests showed that by the use of a content delivery network it is possible to decrease the response times for customers in North America to the same level as they are currently for customers in Europe. The test applications and the content delivery network could have been even more optimised, which would have further improved the performance. These kinds of good results were not the case in all of the testing locations. The results show that especially the results from Hong Kong left a lot to be desired upon. This however does not mean that content delivery networks in general could not provide good performance in Asia, rather it only seems as an area that is not strong for the CloudFront content delivery network. If a consistently strong performance

is needed in Asia, CloudFront might not be the best content delivery network for that. One possible option is also to combine several content delivery networks from different vendors.

The tests showed the importance of using several locations. It cannot be trusted that if a content delivery network performs well in one location that it will also perform well in all of the other locations that it reaches. The other important factor was the length of the break between the tests. It depends on the content delivery network how long the files will approximately stay cached on the edge servers. The best way to test a content delivery network's performance is to use actual real data. In these tests the length of the breaks were fixed length. In a real scenario the breaks are arbitrary. In addition to this, the way that a content delivery network determines the popularity of a file may vary. As the file's popularity plays an important role in the performance, it is also important to understand which files are put into the content delivery network. The performance is best for popular files that are for example used on the Web application's front page, but the performance probably is not as good for files that are only seldom requested.

As shown by the results, the response times decreased in each of the testing locations. Adding more requests to the files made the performance even better. Therefore it is safe to say that the new architecture is an improvement to the old architecture, it is merely a matter of how much better and how expensive will it be.

6 CONCLUSIONS

Scaling up Web architectures is in the end a numbers game and a matter of choosing the right tool for the job. Whenever making decisions about the architecture, the current situation needs to be reviewed and a decision needs to be made on a solution that meets the requirements. One of the biggest problems is finding a good mixture of affordability and quality. In this master's thesis such a solution was found. The approach was sought by looking into the theory and by interviewing an expert. A solution was designed that best met the requirements set by the case environment. The new architecture was globally tested and the test results were analysed.

The first and most important requirement was that of low latency to all users globally. As shown in Chapter 5, the response times significantly lowered in countries faraway from Finland and also provided a low latency in countries that are near to Finland. The solution also meets the other requirements. The Web application can still be served from a single domain name. The data is synchronised in the content delivery network and any data that needs to be synchronised in real time is still kept on the same server, so there is no problem with data consistency. As a content delivery network does create another place to serve content, special care has to be made while updating this content so that it is consistent with the other data related to it however technical workarounds can be made for this so that there is no real problem. The most critical data is still kept on the same Internet hosting service provider and in case of a hardware failure in the content delivery network, there is no data lost, but another server is used automatically. The transitioning phase to the new architecture is easy and users can even be transferred little by little to use the new architecture. The amount of technical changes is also relatively low with only minor changes to the paths where the content is requested. If WordDive wants to use the content delivery network for serving data with restricted access, new security measures may have to be sought.

While the tests showed good results, the tests also showed that the solution does not provide consistently good results in all of the test locations. Before WordDive can adapt to this new architecture, the solution should be further optimised to suit their needs and possibly other content delivery networks should be tested. As Finland still has more WordDive users than any other country and WordDive's current architecture is not in an immediate need of scaling up, one suitable option is to just use the content delivery network for serving the content to users who are not in Finland. This would mean that the content delivery network would cost less, as there would be fewer requests made to it. Finnish users will not gain as much benefits from using the content delivery network as some other users, because the response times are already so good, due to the vicinity

of the clients and the server. When WordDive's current server starts to reach the end of its capacity, the Finnish users can also start to use the content delivery network and so the load on the server will decrease. To know when to scale up, WordDive should continue to measure the load on their server and try to figure out, which things actually cause it. And as always the code and the content should be optimised all the time, as there might be no need for scaling up the architecture, if quick optimisations can be made on these fronts.

This master's thesis solved a common problem among start up Web companies who are aiming to broaden their user base. The designed solution will give WordDive the possibility to provide the same great user experience to users from the other side of the world, due to the reduced latency. In addition to this, the service can now be served more flexibly as the new architecture scales up automatically.

REFERENCES

Allspaw, J. 2008. The Art of Capacity Planning: Scaling Web Resources. O'Reilly Media, Inc. 160 p.

Amazon CloudFront. 2011. [WWW]. [Accessed 5.8.2011]. http://aws.amazon.com/cloudfront/

Buyya, R. et al. 2008. Content Delivery Networks. Springer. 417 p.

Davies, A. & Fisk, H. 2006. MySQL clustering. Sams Publishing. 202 p.

Henderson, C. 2006. Building Scalable Web Sites. O'Reilly. Kindle version. Accessed July 2011 from Amazon.com.

Jain, R. 2006. Summarizing Measured Data. [WWW]. [Accessed 30.8.2011]. http://www.cse. wustl.edu/~jain/cse567-06/ftp/k_12smd.pdf

Mell, P. & Grance, T. 2009. The NIST Definition of Cloud Computing, National Institute of Standards and Technology, Information Technology Laboratory, Version 15.

MySQL. 2011. MySQL Cluster Overview. [WWW]. [Accessed 8.8.2011]. http://dev.mysql.com/doc/refman/5.0/en/mysql-cluster-overview.html

Pallis, G. & Vakali, A. 2006. Insight and Perspectives for Content Delivery Networks, Communications of the ACM, Vol. 49, No. 1, ACM Press, NY, USA, pp. 101-106.

Schlossnagle, T. 2006. Scalable Internet Architectures. Sams. Kindle version. Accessed July 2011 from Amazon.com.

Systä, T. 2010. Short introduction to cloud computing. [WWW]. [Accessed 8.8.2011]. http://www.cs.tut.fi/%7Etsysta/CC_Intro.pdf

Open Compute Project. 2011. Open Compute Project – Hacking Conventional Compitung Infrastructure. [WWW]. [Accessed 8.8.2011]. http://www.opencompute.org/